PROPKA 3

Release 3.4.0+0.g68007f0.dirty

Jan H. Jensen, Chresten R. Søndergaard, Mats H. M. Olsson, Mich

Dec 19, 2020

CONTENTS

Release 3.4.0+0.g68007f0.dirty

Date Dec 19, 2020

PROPKA 3 predicts the pK_a values of ionizable groups in proteins [Sondergaard2011] and protein-ligand complexes based on the 3D structure [Olsson2011].

This package installs the *propka3 command* and the `propka` Python package.

CHAPTER

ONE

LICENSE AND SOURCE CODE

PROPKA 3 is released under the GNU Lesser General Public License v2.1 (see the files *LICENSE* in the repository for details).

Source code is available in the public GitHub repository https://github.com/jensengroup/propka.

CHAPTER

TWO

GETTING HELP

Please report *bugs and feature requests* for PROPKA through the Issue Tracker.

CHAPTER

THREE

CONTRIBUTING

PROPKA welcomes new contributions. To contribute code, submit a *pull request* against the master branch in the propka repository.

CHAPTER

FOUR

CITATION

If you use PROPKA 3 in published work please cite [Sondergaard2011] and [Olsson2011].

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

5.1 Installation

PROPKA 3 requires Python 3.6 or higher. Additional requirements are specified in the `requirements.txt` file and automatically satisfied when installing with pip.

5.1.1 `pip`-based installation

The easiest way to install a release of PROPKA 3 is from the PyPI archive with the command

```
pip install --upgrade propka
```

This installation will install the *propka* Python module and the **propka3** executable script. As always, a virtual environment (e.g., via virtualenv) is recommended when installing packages.

5.1.2 Source-based installation

The source code can be installed by cloning the repository or unpacking from a source code archive and running

```
pip install .
```

in the source directory.

For the purposes of testing or development, you may prefer to install PROPKA as an editable module via pip by running

```
pip install -e .
```

in the source directory.

5.2 Quickstart Guide

PROPKA can be used either via the installed script **propka3** or as a Python module. When using the *propka3 command*, use

```
propka3  FILENAME
```

As a module (*propka*), also provide the input filename

```
python -m propka FILENAME
```

In both cases, additional options may be added, as described in more detail for the *propka3 command*.

5.2.1 Predicting protein residue pK_a values

Most users run PROPKA by invoking the **propka3** program with a PDB file as its argument; e.g., for PDB 1HPX (HIV-1 protease complexed with the inhibitor KNI-272)

```
propka3 1hpx.pdb
```

In this example, pK_a values of titratable protein residues and titratable groups of the inhibitor KNI are calculated.

The output looks similar to the following (many lines omitted as "..."). It is also contained in the output file `1hpx.pka` that is automatically written:

```
propka3.2                                                                      ⌙
 ↪       2020-06-19
...
...
Found NAR group:  1530-  N1   900-KNI (B) [   7.907    1.459    5.427] N
Found O3  group:  1531-  O1   900-KNI (B) [   5.235    3.791    9.082] O
Found O2  group:  1532-  O3   900-KNI (B) [   3.327    4.297   11.852] O
Found NAM group:  1533-  N2   900-KNI (B) [   3.955    2.384   10.893] N
Found O2  group:  1539-  O6   900-KNI (B) [   3.758   -0.629   12.111] O
Found NAM group:  1541-  N3   900-KNI (B) [   4.496    0.982   13.492] N
Found O2  group:  1542-  O4   900-KNI (B) [   6.324   -1.234   17.045] O
Found OH  group:  1548-  O2   900-KNI (B) [   4.949    0.934   16.427] O
Found O2  group:  1559-  O5   900-KNI (B) [   6.746   -3.574   14.588] O
Found NAM group:  1560-  N5   900-KNI (B) [   7.637   -4.575   16.403] N
-------------------------------------------------------------------------------
 ↪-----------------


Calculating pKas for Conformation container 1A with 1878 atoms and 480 groups
-------------------------------------------------------------------------------
 ↪-----------------

---------  -----  ------  ---------------------  --------------  --------------⌙
 ↪   ---------------
                          DESOLVATION  EFFECTS       SIDECHAIN          BACKBONE  ⌙
 ↪    COULOMBIC
 RESIDUE    pKa    BURIED     REGULAR      RE        HYDROGEN BOND     HYDROGEN BOND ⌙
 ↪    INTERACTION
---------  -----  ------  ---------  ---------  --------------  --------------⌙
 ↪   ---------------
```

(continues on next page)

(continued from previous page)

```
ASP  25 A   5.07*  100 %    4.30  617   0.19    0   -0.85 KNI  O4 B   -0.63 GLY  27 A␣
↪   0.07 ASP  29 A
ASP  25 A                                          -0.85 KNI  O2 B   -0.09 ALA  28 A␣
↪   0.00 XXX   0 X
ASP  25 A                                          -0.84 ASP  25 B   -0.04 GLY  27 B␣
↪   0.00 XXX   0 X

ASP  29 A   3.11    50 %    1.20  420   0.13    0   -0.68 ARG  87 A    0.00 XXX   0 X␣
↪  -0.04 LYS  45 A
ASP  29 A                                          -0.28 ARG   8 B    0.00 XXX   0 X␣
↪  -0.47 ARG  87 A
ASP  29 A                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪  -0.54 ARG   8 B

ASP  30 A   4.62    59 %    1.30  446   0.00    0   -0.11 LYS  45 A    0.00 XXX   0 X␣
↪  -0.07 ARG  87 A
ASP  30 A                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪  -0.01 ARG   8 B
ASP  30 A                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪   0.29 ASP  29 A
ASP  30 A                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪  -0.57 LYS  45 A

ASP  60 A   2.55     0 %    0.41  249   0.00    0   -0.40 THR  74 A    0.00 XXX   0 X␣
↪  -0.02 LYS  45 A
ASP  60 A                                          -0.85 LYS  43 A    0.00 XXX   0 X␣
↪  -0.38 LYS  43 A
...
...
...
ARG  87 B  12.28    45 %   -1.40  407   0.00    0    0.77 ASP  29 B    0.00 XXX   0 X␣
↪   0.10 ASP  30 B
ARG  87 B                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪  -0.19 ARG   8 A
ARG  87 B                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪   0.50 ASP  29 B

N+    1 B   8.96     0 %   -0.39  235   0.00    0    0.85 C-   99 A    0.00 XXX   0 X␣
↪   0.07 CYS  67 B
N+    1 B                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪   0.04 CYS  95 B
N+    1 B                                           0.00 XXX   0 X    0.00 XXX   0 X␣
↪   0.38 C-   99 A

KNI  N1 B   4.60     0 %   -0.36  273   0.00    0    0.00 XXX   0 X    0.00 XXX   0 X␣
↪  -0.03 ARG   8 A

Coupled residues (marked *) were detected.Please rerun PropKa with the --display-
↪coupled-residues
or -d option for detailed information.

---------------------------------------------------------------------------------
↪------------------
SUMMARY OF THIS PREDICTION
       Group      pKa  model-pKa   ligand atom-type
   ASP  25 A     5.07       3.80
   ASP  29 A     3.11       3.80
```

(continues on next page)

(continued from previous page)

```
   ASP  30 A     4.62       3.80
   ASP  60 A     2.55       3.80
   ASP  25 B     9.28       3.80
   ASP  29 B     1.78       3.80
   ASP  30 B     4.91       3.80
   ASP  60 B     2.13       3.80
   GLU  21 A     4.78       4.50
   GLU  34 A     3.93       4.50
   GLU  35 A     3.65       4.50
   GLU  65 A     3.89       4.50
   GLU  21 B     4.73       4.50
   GLU  34 B     3.36       4.50
   GLU  35 B     4.07       4.50
   GLU  65 B     3.70       4.50
   C-   99 A     2.08       3.20
   C-   99 B     2.11       3.20
   HIS  69 A     6.98       6.50
   HIS  69 B     7.11       6.50
   CYS  67 A     9.41       9.00
   CYS  95 A    11.68       9.00
   CYS  67 B     9.82       9.00
   CYS  95 B    11.61       9.00
   TYR  59 A     9.67      10.00
   TYR  59 B     9.54      10.00
   LYS  14 A    10.43      10.50
   LYS  20 A    10.32      10.50
   LYS  43 A    11.41      10.50
   LYS  45 A    10.54      10.50
   LYS  55 A    10.42      10.50
   LYS  70 A    10.92      10.50
   LYS  14 B    10.55      10.50
   LYS  20 B    11.01      10.50
   LYS  43 B    11.43      10.50
   LYS  45 B    10.47      10.50
   LYS  55 B    10.41      10.50
   LYS  70 B    11.07      10.50
   ARG   8 A    13.96      12.50
   ARG  41 A    12.41      12.50
   ARG  57 A    14.40      12.50
   ARG  87 A    12.35      12.50
   ARG   8 B    12.76      12.50
   ARG  41 B    12.42      12.50
   ARG  57 B    13.73      12.50
   ARG  87 B    12.28      12.50
   N+    1 A     8.96       8.00
   N+    1 B     8.96       8.00
   KNI  N1 B     4.60       5.00                NAR

Writing 1hpx.pka
```

Some of the important contents:

- The section *Calculating pKas for Conformation container 1A with 1878 atoms and 480 groups* lists details on the calculations for all ionizable residues. It indicates the considerations that went into a pK_a estimate such as hydrogen bonds and Coulomb interactions. It also indicates if there is potentially coupling between residues.
- Values with "XXX" placeholders are not calculated (but appear to maintain the formatting).

- The section *SUMMARY OF THIS PREDICTION* lists the predicted pK_a for each residue together with the model pK_a (the "default" value).
- Ligand values are labeled with the residue name of the ligand, in this case "KNI".

5.2.2 Getting help

A brief list of available options can be obtained by running PROPKA with no options. A longer list of options and descriptions is available using the *propka3 --help* option:

```
propka3 --help
```

5.3 propka3 command

PROPKA predicts the pKa values of ionizable groups in proteins and protein-ligand complexes based in the 3D structure. The **propka3** command has the following options:

```
propka3 [-h] [-f FILENAMES] [-r REFERENCE] [-c CHAINS] [-i TITRATE_ONLY] [-t␣
↪THERMOPHILES] [-a ALIGNMENT] [-m MUTATIONS]
       [-v VERSION_LABEL] [-p PARAMETERS] [--log-level {DEBUG,INFO,WARNING,ERROR,
↪CRITICAL}] [-o PH] [-w WINDOW WINDOW WINDOW]
       [-g GRID GRID GRID] [--mutator MUTATOR] [--mutator-option MUTATOR_OPTIONS] [-
↪d] [-l] [-k] [-q] [--protonate-all]
       input_pdb
```

input_pdb
read data from file <input_pdb>

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

-f FILENAMES, **--file** FILENAMES
read data from <filename>, i.e. <filename> is added to arguments (default: [])

-r REFERENCE, **--reference** REFERENCE
setting which reference to use for stability calculations [neutral/low-pH] (default: neutral)

-c CHAINS, **--chain** CHAINS
creating the protein with only a specified chain. Specify " " for chains without ID [all] (default: None)

-i TITRATE_ONLY, **--titrate_only** TITRATE_ONLY
Treat only the specified residues as titratable. Value should be a comma-separated list of "chain:resnum" values; for example: `-i "A:10,A:11"` (default: None)

-t THERMOPHILES, **--thermophile** THERMOPHILES
defining a thermophile filename; usually used in 'alignment-mutations' (default: None)

-a ALIGNMENT, **--alignment** ALIGNMENT
alignment file connecting <filename> and <thermophile> [<thermophile>.pir] (default: None)

-m MUTATIONS, **--mutation** MUTATIONS
specifying mutation labels which is used to modify <filename> according to, e.g. N25R/N181D (default: None)

--version
show program's version number and exit

-p PARAMETERS, **--parameters** PARAMETERS
set the parameter file (default: <installation_directory>/propka/propka/propka.cfg)

--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}
logging level verbosity (default: INFO)

-o PH, **--pH** PH
setting pH-value used in e.g. stability calculations (default: 7.0)

-w WINDOW WINDOW WINDOW, **--window** WINDOW WINDOW WINDOW
setting the pH-window to show e.g. stability profiles (default: (0.0, 14.0, 1.0))

-g GRID GRID GRID, **--grid** GRID GRID GRID
setting the pH-grid to calculate e.g. stability related properties (default: (0.0, 14.0, 0.1))

--mutator MUTATOR
setting approach for mutating <filename> [alignment/scwrl/jackal] (default: None)

--mutator-option MUTATOR_OPTIONS
setting property for mutator [e.g. type="side-chain"] (default: None)

-d, --display-coupled-residues
Displays alternative pKa values due to coupling of titratable groups (default: False)

-l, --reuse-ligand-mol2-files
Reuses the ligand mol2 files allowing the user to alter ligand bond orders (default: False)

-k, --keep-protons
Keep protons in input file (default: False)

-q, --quiet
suppress non-warning messages (default: None)

--protonate-all
Protonate all atoms (will not influence pKa calculation) (default: False)

5.4 API Reference

The **propka3** command provides a command-line interface to PROPKA 3's functionality. It is built on classes and functions in the *propka* module. The API of *propka* is documented here for developers who might want to directly use the PROPKA 3 code.

Note: The API is still changing and there is currently no guarantee that it will remain stable between minor releases.

5.4.1 Data structures

atom	Atom
bonds	Bonds
group	Data structures for groups
conformation_container	Molecular data structures
molecular_container	PDB molecular container

propka.atom

Atom

The *Atom* class contains all atom information found in the PDB file.

Classes

Atom([line])	Atom class - contains all atom information found in the PDB file

class propka.atom.**Atom**(*line=None*)

Atom class - contains all atom information found in the PDB file

Changed in version 3.4.0: `make_input_line()` and `get_input_parameters()` have been removed as reading/writing PROPKA input is no longer supported.

count_bonded_elements(*element*)

Count number of bonded atoms with same element.

Parameters **element** – element type for test.

Returns number of bonded atoms.

get_bonded_elements(*element*)

Get bonded atoms with same element.

Parameters **element** – element type for test.

Returns array of bonded atoms.

get_bonded_heavy_atoms()

Get the atoms bonded to this one that aren't hydrogen.

Returns list of atoms.

get_tidy_label()

Returns a 'tidier' atom label for printing the new pdbfile

TODO - this could/should be a @property method/attribute

Returns String with label

is_atom_within_bond_distance(*other_atom*, *max_bonds*, *cur_bond*)

Check if <other_atom> is found within <max_bonds> bonds of self.

Parameters

- **other_atom** – atom to check
- **max_bonds** – number of bonds to check for other atom bonding to self

Returns Boolean for atom bond distance

make_conect_line()

PDB line for bonding within this molecule.

Returns String with PDB line.

make_copy()

Make a copy of this atom.

Returns Another atom object copy of this one.

make_mol2_line(*id_*)
Create MOL2 line.

Format: 1 S1 3.6147 2.0531 1.4795 S.3 1 noname -0.1785

TODO - this could/should be a @property method/attribute

Returns String with MOL2 line.

make_pdb_line()
Create PDB line.

TODO - this could/should be a @property method/attribute TODO - figure out difference between make_pdb_line, and make_pdb_line2

Returns String with PDB line.

make_pdb_line2(*numb=None*, *name=None*, *res_name=None*, *chain_id=None*, *res_num=None*, *x=None*, *y=None*, *z=None*, *occ=None*, *beta=None*)
Create a PDB line.

TODO - this could/should be a @property method/attribute TODO - figure out difference between make_pdb_line, and make_pdb_line2

Returns String with PDB line.

set_group_type(*type_*)
Set group type of atom.

Parameters **type** – group type of atom

set_properties(*line*)
Line from PDB file to set properties of atom.

Parameters **line** – PDB file line

set_property(*numb=None*, *name=None*, *res_name=None*, *chain_id=None*, *res_num=None*, *x=None*, *y=None*, *z=None*, *occ=None*, *beta=None*)
Set properties of the atom object.

Parameters

- **numb** – Atom number
- **name** – Atom name
- **res_name** – residue name
- **chain_id** – chain ID
- **res_num** – residue number
- **x** – atom x-coordinate
- **y** – atom y-coordinate
- **z** – atom z-coordinate
- **occ** – atom occupancy
- **beta** – atom temperature factor

set_residue(*residue*)
Makes a reference to the parent residue

Parameters **residue** – the parent residue

propka.bonds

Bonds

PROPKA representation of bonds.

Classes

BondMaker()	Makes bonds?

class propka.bonds.**BondMaker**
Makes bonds?

TODO - the documentation for this class needs to be improved.

add_pi_electron_information(*molecules*)
Add pi electron information to a molecule.

Parameters **molecules** – list of molecules for adding pi electron information.

add_pi_electron_table_info(*atoms*)
Add table information on pi electrons

Parameters **atoms** – list of atoms for pi electron table information checking

check_distance(*atom1*, *atom2*)
Check distance between two atoms

Parameters

- **atom1** – first atom for distance check
- **atom2** – second atom for distance check

Returns True if within distance, False otherwise

check_for_cysteine_bonds(*cys1*, *cys2*)
Looks for potential bonds between two cysteines.

Parameters

- **cys1** – one of the cysteines to check
- **cys1** – one of the cysteines to check

connect_backbone(*residue1*, *residue2*)
Sets up bonds in the backbone

Parameters

- **residue1** – first residue to connect
- **residue2** – second residue to connect

find_bonds_for_atoms(*atoms*)
Finds all bonds for a list of atoms

Parameters **atoms** – list of atoms in which to find bonds.

find_bonds_for_atoms_disjoint(*atoms1*, *atoms2*)
Finds all bonds between two disjoint sets of atoms.

Parameters

- **atoms1** – list of atoms
- **atoms2** – list of atoms

find_bonds_for_atoms_using_boxes(*atoms*)
Finds all bonds for a list of atoms.

Parameters **atoms** – list of atoms for finding bonds

find_bonds_for_ligand(*ligand*)
Finds bonds for all atoms in the ligand molecule

Parameters **ligand** – ligand molecule to search for bonds

find_bonds_for_molecules_using_boxes(*molecules*)
Finds all bonds for a molecular container.

Parameters **molecules** – list of molecules for finding bonds.

find_bonds_for_protein(*protein*)
Bonds proteins based on the way atoms normally bond.

Parameters **protein** – the protein to search for bonds

find_bonds_for_protein_by_distance(*molecule*)
Finds bonds for all atoms in the molecule.

Parameters **molecule** – molecule in which to find bonds.

Returns list of atoms

find_bonds_for_residue_backbone(*residue*)
Find bonds for this residue's backbone.

Parameters **residue** – reside to search for backbone bonds.

find_bonds_for_side_chain(*atoms*)
Finds bonds for a side chain.

Parameters **atoms** – list of atoms to check for bonds

find_bonds_for_terminal_oxygen(*residue*)
Look for bonds for terminal oxygen.

Parameters **- test residue**(*residue*) –

generate_protein_bond_dictionary(*atoms*)
Generate dictionary of protein bonds.

Parameters **atoms** – list of atoms for bonding

static has_bond(*atom1*, *atom2*)
Look for bond between two atoms.

Parameters

- **atom1** – first atom to check
- **atom2** – second atom to check

Returns True if there is a bond between atoms

static make_bond(*atom1*, *atom2*)
Makes a bond between atom1 and atom2

Parameters

- **atom1** – first atom to bond

- **atom2** – second atom to bond

propka.group

Data structures for groups

Routines and classes for storing groups important to PROPKA calculations.

Changed in version 3.4.0: Removed `initialize_atom_group()` as reading PROPKA inputs is no longer supported.

Module Attributes

EXPECTED_ATOMS_ACID_INTERACTIONS	acids
EXPECTED_ATOMS_BASE_INTERACTIONS	bases

Functions

is_group(parameters, atom)	Identify whether the atom belongs to a group.
is_ion_group(parameters, atom)	Identify whether the atom belongs to an ion group.
is_ligand_group_by_groups(_, atom)	Identify whether the atom belongs to a ligand group by checking groups.
is_ligand_group_by_marvin_pkas(parameters, atom)	Identify whether the atom belongs to a ligand group by calculating 'Marvin pKas'.
is_protein_group(parameters, atom)	Identify whether the atom belongs to a protein group.

Classes

AMDGroup(atom)	Amide group.
ARGGroup(atom)	Arginine group.
BBCGroup(atom)	Backbone carbon group.
BBNGroup(atom)	Backbone nitrogen group.
C2NGroup(atom)	Amidinium group.
CGGroup(atom)	Guadinium group.
COOGroup(atom)	Carboxyl group.
CYSGroup(atom)	Cysteine group.
ClGroup(atom)	Chloride group.
CtermGroup(atom)	C-terminus group.
FGroup(atom)	Fluoride group.
Group(atom)	Class for storing groups important to pKa calculations.
HISGroup(atom)	Histidine group.
IonGroup(atom)	Ion group.
LYSGroup(atom)	Lysine group.
N1Group(atom)	Unknown group.
N30Group(atom)	Unknown group.
N31Group(atom)	Unknown group.
N32Group(atom)	Unknown group.

continues on next page

Table 6 – continued from previous page

N33Group(atom)	Unknown group.
NAMGroup(atom)	Unknown group.
NARGroup(atom)	Unknown group.
NP1Group(atom)	Unknown group.
NonTitratableLigandGroup(atom)	Non-titratable ligand group.
NtermGroup(atom)	N-terminus group.
O2Group(atom)	Unknown group.
O3Group(atom)	Unknown group.
OCOGroup(atom)	Carboxyl group.
OHGroup(atom)	Hydroxide group.
OPGroup(atom)	Phosphate group.
ROHGroup(atom)	Alcohol group.
SERGroup(atom)	Serine group.
SHGroup(atom)	Sulfhydryl group.
TRPGroup(atom)	Tryptophan group.
TYRGroup(atom)	Tyrosine group.
TitratableLigandGroup(atom)	Titratable ligand group.

class propka.group.**AMDGroup**(*atom*)
Amide group.

setup_atoms()
Setup group.

class propka.group.**ARGGroup**(*atom*)
Arginine group.

setup_atoms()
Set up group.

class propka.group.**BBCGroup**(*atom*)
Backbone carbon group.

setup_atoms()
Set up atoms in group.

class propka.group.**BBNGroup**(*atom*)
Backbone nitrogen group.

setup_atoms()
Set up atoms in group.

class propka.group.**C2NGroup**(*atom*)
Amidinium group.

setup_atoms()
Set up atoms in this group.

class propka.group.**CGGroup**(*atom*)
Guadinium group.

setup_atoms()
Set up atoms in this group.

class propka.group.**COOGroup**(*atom*)
Carboxyl group.

setup_atoms()
Set up group.

class propka.group.**CYSGroup**(*atom*)
Cysteine group.

class propka.group.**ClGroup**(*atom*)
Chloride group.

class propka.group.**CtermGroup**(*atom*)
C-terminus group.

setup_atoms()
Set up atoms in group.

propka.group.**EXPECTED_ATOMS_ACID_INTERACTIONS = {'AMD': {'H': 2, 'N': 1}, 'ARG': {'H': 5,**
acids

propka.group.**EXPECTED_ATOMS_BASE_INTERACTIONS = {'AMD': {'O': 1}, 'ARG': {'N': 3}, 'BBC':**
bases

class propka.group.**FGroup**(*atom*)
Fluoride group.

class propka.group.**Group**(*atom*)
Class for storing groups important to pKa calculations.

Changed in version 3.4.0: Removed `make_covalently_coupled_line()` and `make_non_covalently_coupled_line()` as writing PROPKA inputs is no longer supported.

add_determinant(*new_determinant*, *type_*)
Add to current and creates non-present determinants.

Parameters

- **new_determinant** – new determinant to add
- **type** – determinant type

calculate_charge(*_*, *ph=7.0*, *state='folded'*)
Calculate the charge of the specified state at the specified pH.

Parameters

- **_** – parameters for calculation
- **ph** – pH value
- **state** – "folded" or "unfolded"

Returns float with charge

calculate_folding_energy(*parameters*, *ph=None*, *reference=None*)
Return the electrostatic energy of this residue at specified pH.

Parameters

- **parameters** – parameters for energy calculation
- **ph** – pH value for calculation
- **reference** – reference state for calculation

Returns float describing energy

calculate_intrinsic_pka()
Calculate the intrinsic pKa values from the desolvation determinants, back-bone hydrogen bonds, and side-chain hydrogen bonds to non-titratable residues.

calculate_total_pka()
Calculate total pKa based on determinants associated with this group.

clone()
Create a copy of this group.

Returns Copy of this group.

couple_covalently(*other*)
Couple this group with another group.

Parameters **other** – other group for coupling

couple_non_covalently(*other*)
Non-covalenthly couple this group with another group.

Parameters **other** – other group for coupling

get_covalently_coupled_groups()
Get covalently coupled groups.

Returns list of covalently coupled groups.

get_determinant_for_string(*type_*, *number*)
Return a string describing determinant.

Parameters

- **type** – determinant type
- **number** – determinant index number

Returns string

get_determinant_string(*remove_penalised_group=False*)
Create a string to identify this determinant.

Parameters **remove_penalised_group** – Boolean flag to remove penalized groups

Returns string

get_interaction_atoms(*interacting_group*)
Get atoms involved in interaction with other group.

Parameters **interacting_group** – other group

Returns list of atoms

get_non_covalently_coupled_groups()
Get non-covalently coupled groups.

Returns list of covalently coupled groups.

get_summary_string(*remove_penalised_group=False*)
Create summary string for this group.

Parameters **remove_penalised_group** – Boolean to ignore penalized groups

Returns string

remove_determinants(*labels*)
Remove all determinants with specified labels.

Parameters **labels** – list of labels to remove

set_center(*atoms*)
Set center of group based on atoms.

Parameters **atoms** – list of atoms

set_determinant(*new_determinant*, *type_*)
Overwrite current and create non-present determinants.

Parameters

- **new_determinant** – new determinant to add
- **type** – determinant type

set_interaction_atoms(*interaction_atoms_for_acids*, *interaction_atoms_for_bases*)
Set interacting atoms and group types.

Parameters

- **interaction_atoms_for_acids** – list of atoms for acid interactions
- **interaction_atoms_for_base** – list of atoms for base interactions

setup()
Set up a group.

setup_atoms()
Set up atoms in group.

This method is overwritten for some types of groups

share_determinant(*new_determinant*, *type_*)
Add determinant to this group's list of determinants.

Parameters

- **new_determinant** – determinant to add
- **type** – type of determinant

share_determinants(*others*)
Share determinants between this group and others.

Parameters **others** – list of other groups

use_in_calculations()
Indicate whether group should be included in results report.

If –titrate_only option is specified, only residues that are titratable and are in that list are included; otherwise all titratable residues and CYS residues are included.

class propka.group.**HISGroup**(*atom*)
Histidine group.

setup_atoms()
Set up atoms in group.

class propka.group.**IonGroup**(*atom*)
Ion group.

class propka.group.**LYSGroup**(*atom*)
Lysine group.

class propka.group.**N1Group**(*atom*)
Unknown group.

TODO - identify this group.

class propka.group.**N30Group**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in this group.

class propka.group.**N31Group**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in this group.

class propka.group.**N32Group**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in this group.

class propka.group.**N33Group**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in this group.

class propka.group.**NAMGroup**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in this group.

class propka.group.**NARGroup**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in group.

class propka.group.**NP1Group**(*atom*)
Unknown group.

TODO - identify this group.

setup_atoms()
Set up atoms in group.

class propka.group.**NonTitratableLigandGroup**(*atom*)
Non-titratable ligand group.

class propka.group.**NtermGroup**(*atom*)
N-terminus group.

class propka.group.**O2Group**(*atom*)
Unknown group.

TODO - identify this group.

class propka.group.**O3Group**(*atom*)
Unknown group.

TODO - identify this group.

class propka.group.**OCOGroup**(*atom*)
Carboxyl group.

setup_atoms()
Set up atoms in group.

class propka.group.**OHGroup**(*atom*)
Hydroxide group.

setup_atoms()
Set up atoms in this group.

class propka.group.**OPGroup**(*atom*)
Phosphate group.

setup_atoms()
Set up atoms in this group.

class propka.group.**ROHGroup**(*atom*)
Alcohol group.

class propka.group.**SERGroup**(*atom*)
Serine group.

class propka.group.**SHGroup**(*atom*)
Sulfhydryl group.

class propka.group.**TRPGroup**(*atom*)
Tryptophan group.

setup_atoms()
Set up atoms in group.

class propka.group.**TYRGroup**(*atom*)
Tyrosine group.

class propka.group.**TitratableLigandGroup**(*atom*)
Titratable ligand group.

propka.group.**is_group**(*parameters*, *atom*)
Identify whether the atom belongs to a group.

Parameters

- **parameters** – parameters for check
- **atom** – atom to check

Returns group for atom or None

propka.group.**is_ion_group**(*parameters*, *atom*)
Identify whether the atom belongs to an ion group.

Parameters

- **parameters** – parameters for check
- **atom** – atom to check

Returns group for atom or None

propka.group.**is_ligand_group_by_groups**(_, *atom*)
Identify whether the atom belongs to a ligand group by checking groups.

Parameters

- _ – parameters for check
- **atom** – atom to check

Returns group for atom or None

propka.group.**is_ligand_group_by_marvin_pkas**(*parameters*, *atom*)
Identify whether the atom belongs to a ligand group by calculating 'Marvin pKas'.

Parameters

- **parameters** – parameters for check
- **atom** – atom to check

Returns group for atom or None

propka.group.**is_protein_group**(*parameters*, *atom*)
Identify whether the atom belongs to a protein group.

Parameters

- **parameters** – parameters for check
- **atom** – atom to check

Returns group for atom or None

propka.conformation_container

Molecular data structures

Container data structure for molecular conformations.

Module Attributes

UNICODE_MULTIPLIER	A large number that gets multipled with the integer obtained from applying `ord()` to the atom chain ID.
RESIDUE_MULTIPLIER	A number that gets mutiplied with an atom's residue number.

Classes

ConformationContainer([name, parameters, ...])	Container for molecular conformations

class propka.conformation_container.**ConformationContainer**(*name=''*, *parameters=None*, *molecular_container=None*)
Container for molecular conformations

Changed in version 3.4.0: Removed `additional_setup_when_reading_input_files()` as reading

PROPKA inputs is no longer supported.

add_atom(*atom*)
Add atom to container.

Parameters **atom** – atom to add

calculate_charge(*parameters*, *ph=None*)
Calculate charge for folded and unfolded states.

Parameters

- **parameters** – parameters for calculation
- **ph** – pH for calculation

Returns

1. charge for unfolded state
2. charge for folded state

calculate_folding_energy(*ph=None*, *reference=None*)
Calculate folding energy over all groups in conformation container.

Parameters

- **ph** – pH for calculation
- **reference** – reference state

Returns folding energy TODO - need units

calculate_pka(*version*, *options*)
Calculate pKas for conformation container.

Parameters

- **version** – version object
- **options** – option object

copy_atom(*atom*)
Add a copy of the atom to container.

Parameters **atom** – atom to copy and add

coupling_effects()
Penalize groups based on coupling effects.

Bases: The group with the highest pKa (the most stable one in the charged form) will be the first one to adopt a proton as pH is lowered and this group is allowed to titrate. The remaining groups are penalised.

Acids: The group with the highest pKa (the least stable one in the charged form) will be the last group to loose the proton as pH is raised and will be penalised. The remaining groups are allowed to titrate.

extract_groups()
Generate molecular groups needed for calculating pKa values.

find_bonded_titratable_groups(*atom*, *num_bonds*, *original_atom*)
Find bonded titrable groups.

Parameters

- **atom** – atom to check for bonds
- **num_bonds** – number of bonds for coupling

- **original_atom** – another atom to check for bonds

Returns a set of bonded atom groups

find_covalently_coupled_groups()
Find covalently coupled groups and set common charge centres.

find_group(*group*)
Find a group in the container.

Parameters **group** – group to find

Returns False (if group not found) or group

find_non_covalently_coupled_groups(*verbose=False*)
Find non-covalently coupled groups and set common charge centres.

Parameters **verbose** – verbose output

get_a_coupled_system_of_groups(*new_group*, *coupled_groups*, *get_coupled_groups*)
Set up coupled systems of groups.

Parameters

- **new_group** – added to coupled_groups
- **coupled_groups** – existing coupled groups
- **get_coupled_groups** – TODO - I don't know what this

get_acids()
Get acid groups needed for pKa calculations.

Returns list of groups

get_backbone_co_groups()
Get CO backbone groups needed for pKa calculations.

Returns list of groups

get_backbone_groups()
Get backbone groups needed for the pKa calculations.

Returns list of groups

get_backbone_nh_groups()
Get NH backbone groups needed for pKa calculations.

Returns list of groups

get_backbone_reorganisation_groups()
Get groups involved with backbone reorganization.

Returns list of groups

get_chain(*chain*)
Get atoms associated with a specific chain.

Parameters **chain** – chain to select

Returns list of atoms

get_coupled_systems(*groups*, *get_coupled_groups*)
A generator that yields covalently coupled systems.

Parameters

- **groups** – groups for generating coupled systems

- **get_coupled_groups** – TODO - I don't know what this is

Yields covalently coupled systems

get_covalently_coupled_groups()
Get covalently coupled groups needed for pKa calculations.

Returns list of groups

get_group_names(*group_list*)
Get names of groups in list.

Parameters **group_list** – list to check

Returns list of groups

get_groups_for_calculations()
Get a list of groups that should be included in results report.

If –titrate_only option is specified, only residues that are titratable and are in that list are included; otherwise all titratable residues and CYS residues are included.

Returns list of groups

get_groups_in_residue(*residue*)
Get residue groups needed for pKa calculations.

Parameters **residue** – specific residue with groups

Returns list of groups

get_heavy_ligand_atoms()
Get heavy atoms associated with ligands.

Returns list of atoms

get_ions()
Get ion groups.

Returns list of groups

get_ligand_atoms()
Get atoms associated with ligands.

Returns list of atoms

get_non_covalently_coupled_groups()
Get non-covalently coupled groups needed for pKa calculations.

Returns list of groups

get_non_hydrogen_atoms()
Get atoms that are not hydrogens.

Returns list of atoms

get_sidechain_groups()
Get sidechain groups needed for the pKa calculations.

Returns list of groups

get_titratable_groups()
Get all titratable groups needed for pKa calculations.

Returns list of groups

init_group(*group*)
Initialize the given Group object.

Parameters **group** – group object to initialize

set_common_charge_centres()
Assign charge centers to groups.

set_ligand_atom_names()
Set names for atoms in ligands.

setup_and_add_group(*group*)
Check if we want to include this group in the calculations.

Parameters **group** – group to check

static share_determinants(*groups*)
Share sidechain, backbone, and Coloumb determinants between groups.

Parameters **groups** – groups to share between

sort_atoms()
Sort atoms by *self.sort_atoms_key()* and renumber.

static sort_atoms_key(*atom*)
Generate key for atom sorting.

Parameters **atom** – atom for key generation.

Returns key for atom

top_up(*other*)
Adds any atoms found in *other* but not in this container.

Tops up self with all atoms found in other but not in self.

Parameters **other** – conformation container with atoms to add

propka.conformation_container.**RESIDUE_MULTIPLIER = 1000**
A number that gets mutiplied with an atom's residue number. Used in calculating keys for atom sorting.

propka.conformation_container.**UNICODE_MULTIPLIER = 10000000.0**
A large number that gets multipled with the integer obtained from applying `ord()` to the atom chain ID. Used in calculating atom keys for sorting.

propka.molecular_container

PDB molecular container

Molecular container for storing all contents of PDB files.

Classes

MolecularContainer(parameters[, options])	Container for storing molecular contents of PDB files.

class propka.molecular_container.**MolecularContainer**(*parameters*, *options=None*)

Container for storing molecular contents of PDB files.

TODO - this class name does not conform to PEP8 but has external use. We should deprecate and change eventually.

Changed in version 3.4.0: Removed `write_propka()` and `additional_setup_when_reading_input_file()` as reading and writing PROPKA input files is no longer supported.

average_of_conformations()

Generate an average of conformations.

calculate_pka()

Calculate pKa values.

extract_groups()

Identify the groups needed for pKa calculation.

find_covalently_coupled_groups()

Find covalently coupled groups.

find_non_covalently_coupled_groups()

Find non-covalently coupled groups.

get_charge_profile(*conformation='AVR'*, *grid=[0.0, 14.0, 0.1]*)

Get charge profile for conformation as function of pH.

Parameters

- **conformation** – conformation to test
- **grid** – grid of pH values [min, max, step]

Returns list of charge state values

get_folding_profile(*conformation='AVR'*, *reference='neutral'*, *grid=[0.0, 14.0, 0.1]*)

Get a folding profile.

Parameters

- **conformation** – conformation to select
- **reference** – reference state
- **direction** – folding direction (folding)
- **grid** – the grid of pH values [min, max, step_size]
- **options** – options object

Returns TODO - figure out what these are 1. profile 2. opt 3. range_80pct 4. stability_range

get_pi(*conformation='AVR'*, *grid=[0.0, 14.0, 1]*, *iteration=0*)

Get the isoelectric points for folded and unfolded states.

Parameters

- **conformation** – conformation to test
- **grid** – grid of pH values [min, max, step]

- **iteration** – iteration number of process

Returns

1. Folded state PI
2. Unfolded state PI

top_up_conformations()
Makes sure that all atoms are present in all conformations.

write_pka(*filename=None*, *reference='neutral'*, *direction='folding'*, *options=None*)
Write pKa information to a file.

Parameters

- **filename** – file to write to
- **reference** – reference state
- **direction** – folding vs. unfolding
- **options** – options object

5.4.2 I/O

input	Input handling
lib	Set-up of a PROPKA calculation
output	Output
parameters	Configuration file parameters
hybrid36	Hybrid36 PDB-like file format
ligand_pka_values	Ligand pKa values from Marvin
run	Script functionality
version	Version-based configuration

propka.input

Input handling

Input routines.

Changed in version 3.4.0: Methods to read PROPKA input files (`read_propka()` and `get_atom_lines_from_input()`) have been removed.

Functions

conformation_sorter(conf)	TODO - figure out what this function does.
get_atom_lines_from_pdb(pdb_file[, ...])	Get atom lines from PDB file.
open_file_for_reading(input_file)	Open file or file-like stream for reading.
read_molecule_file(filename, mol_container)	Read input file or stream (PDB or PROPKA) for a molecular container
read_parameter_file(input_file, parameters)	Read a parameter file.
read_pdb(pdb_file, parameters, molecule)	Parse a PDB file.

propka.input.**conformation_sorter**(*conf*)
TODO - figure out what this function does.

propka.input.**get_atom_lines_from_pdb**(*pdb_file*, *ignore_residues=[]*, *keep_protons=False*, *tags=['ATOM ', 'HETATM']*, *chains=None*)
Get atom lines from PDB file.

Parameters

- **pdb_file** – PDB file to parse
- **ignore_residues** – list of residues to ignore
- **keep_protons** – bool to keep/ignore protons
- **tags** – tags of lines that include atoms
- **chains** – list of chains

propka.input.**open_file_for_reading**(*input_file*)
Open file or file-like stream for reading.

TODO - convert this to a context manager

Parameters

- **input_file** – path to file or file-like object. If file-like object,
- **will attempt seek** (*then*) –

propka.input.**read_molecule_file**(*filename: str*, *mol_container*, *stream=None*)
Read input file or stream (PDB or PROPKA) for a molecular container

Parameters

- **filename** (*str*) – name of input file. If not using a filestream via the `stream` argument, should be a path to the file to be read.
- **mol_container** – *MolecularContainer* object.
- **stream** – optional filestream handle. If `None`, then open `filename` as a local file for reading.

Returns updated *MolecularContainer* object.

Raises **ValuError** – if invalid input given

Examples

There are two main cases for using `read_molecule_file`. The first (and most common) is to pass the input file (`filename`) as a string which gives the path of the molecule file to be read (here we also pass a *MolecularContainer* object named `mol_container`).

```
>>> read_molecule_file('test.pdb', mol_container)
<propka.molecular_container.MolecularContainer at 0x7f6e0c8f2310>
```

The other use case is when passing a file-like object, e.g. a `io.StringIO` class, instance. This is done by passing the object via the `stream` argument. Since file-like objects do not usually have an associated file name, an appropirate file name should be passed to the `filename` argument. In this case, `filename` is not opened for reading, but instead is used to help recognise the file type (based on the extension being *.pdb*) and also uses that given `filename` to assign a name to the input *MolecularContainer* object.

```
>>> read_molecule_file('test.pdb', mol_container,
                       stream=string_io_object)
<propka.molecular_container.MolecularContainer at 0x7f6e0c8f2310>
```

Changed in version 3.4.0: PROPKA input files (extension: *.propka_input*) are no longer read.

propka.input.**read_parameter_file**(*input_file*, *parameters*)

Read a parameter file.

Parameters

- **input_file** – input file to read
- **parameters** – Parameters object

Returns updated Parameters object

propka.input.**read_pdb**(*pdb_file*, *parameters*, *molecule*)

Parse a PDB file.

Parameters

- **pdb_file** – file to read
- **parameters** – parameters to guide parsing
- **molecule** – molecular container

Returns

1. list of conformations
2. list of names

Return type list with elements

propka.lib

Set-up of a PROPKA calculation

Implements many of the main functions used to call PROPKA.

Functions

build_parser([parser])	Build an argument parser for PROPKA.
configuration_compare(conf)	TODO - figure out what this function does.
generate_combinations(interactions)	Generate combinations of interactions.
get_sorted_configurations(configuration_keys)	Extract and sort configurations.
loadOptions([args])	Load the arguments parser with options.
make_combination(combis, interaction)	Make a specific set of combinations.
make_grid(min_, max_, step)	Make a grid across the specified tange.
make_molecule(atom, atoms)	Make a molecule from atoms.
make_tidy_atom_label(name, element)	Returns a 'tidier' atom label for printing to the new PDB file.
parse_res_string(res_str)	Parse a residue string.
protein_precheck(conformations, names)	Check protein for correct number of atoms, etc.
resid_from_atom(atom)	Return string with atom residue information.

continues on next page

Table 12 – continued from previous page

split_atoms_into_molecules(atoms)	Maps atoms into molecules.

propka.lib.**build_parser**(*parser=None*)
Build an argument parser for PROPKA.

Parameters **parser** – existing parser. If this is not None, then the PROPKA parser will be created as a subparser to this existing parser. Otherwise, a new parser will be created.

Returns ArgumentParser object.

Changed in version 3.4.0: Argument *–generate-propka-input* has been removed as writing PROPKA input files is no longer supported.

propka.lib.**configuration_compare**(*conf*)
TODO - figure out what this function does.

propka.lib.**generate_combinations**(*interactions*)
Generate combinations of interactions.

Parameters **interactions** – list of interactions

Returns list of combinations

propka.lib.**get_sorted_configurations**(*configuration_keys*)
Extract and sort configurations.

Parameters **configuration_keys** – list of configuration keys

Returns list of configurations

propka.lib.**loadOptions**(*args=None*)
Load the arguments parser with options. Note that verbosity is set as soon as this function is invoked.

Parameters **args** – list of arguments

Returns argparse namespace

propka.lib.**make_combination**(*combis*, *interaction*)
Make a specific set of combinations.

Parameters

- **combis** – list of combinations
- **interaction** – interaction to add to combinations

Returns list of combinations

propka.lib.**make_grid**(*min_*, *max_*, *step*)
Make a grid across the specified tange.

TODO - figure out if this duplicates existing generators like *range* or numpy function.

Parameters

- **min** – minimum value of grid
- **max** – maximum value of grid
- **step** – grid step size

propka.lib.**make_molecule**(*atom*, *atoms*)
Make a molecule from atoms.

Parameters

- **atom** – one of the atoms
- **atoms** – a list of the remaining atoms

Returns list of atoms

propka.lib.**make_tidy_atom_label**(*name*, *element*)
Returns a 'tidier' atom label for printing to the new PDB file.

Parameters

- **name** – atom name
- **element** – atom element

Returns string

propka.lib.**parse_res_string**(*res_str*)
Parse a residue string.

Parameters **res_string** – residue string in format "chain:resnum[inscode]"

Returns a tuple of (chain, resnum, inscode).

Raises **ValueError if the input string is invalid.** –

propka.lib.**protein_precheck**(*conformations*, *names*)
Check protein for correct number of atoms, etc.

Parameters **names** – conformation names to check

propka.lib.**resid_from_atom**(*atom*)
Return string with atom residue information.

Parameters **atom** – atom to generate string for

Returns string

propka.lib.**split_atoms_into_molecules**(*atoms*)
Maps atoms into molecules.

Parameters **atoms** – list of atoms

Returns list of molecules

propka.output

Output

Output routines.

Functions

get_bond_order(atom1, atom2)	Get the order of a bond between two atoms.
get_charge_profile_section(protein[, ...])	Returns string with the charge profile section of the results.
get_determinant_section(protein, ...)	Returns string with determinant section of results.
get_determinants_header()	Create the Determinant header.
get_folding_profile_section(protein[, ...])	Returns string with the folding profile section of the results.
get_propka_header()	Create the header.
get_references_header()	Create the 'references' part of output file.
get_summary_header()	Create the summary header.
get_summary_section(protein, conformation, ...)	Returns string with summary section of the results.
get_the_line()	Draw the line-Johnny Cash would have been proud-or actually Aerosmith!
get_warning_header()	Create the 'warning' part of the output file.
make_interaction_map(name, list_, interaction)	Print out an interaction map named 'name' of the groups in 'list' based on the function 'interaction'
open_file_for_writing(input_file)	Open file or file-like stream for writing.
print_header()	Print header section of output.
print_pka_section(protein, conformation, ...)	Prints out pKa section of results.
print_result(protein, conformation, parameters)	Prints all resulting output from determinants and down.
print_tm_profile(protein[, reference, ...])	Print Tm profile.
write_file(filename, lines)	Writes a new file.
write_jackal_scap_file([mutation_data, ...])	Write a scap file for, i.e., generating a mutated protein
write_mol2_for_atoms(atoms, filename)	Write out MOL2 file for atoms.
write_pdb_for_atoms(atoms, filename[, ...])	Write out PDB file for atoms.
write_pdb_for_conformation(conformation, ...)	Write PDB conformation to a file.
write_pdb_for_protein(protein[, pdbfile, ...])	Write a residue to the new PDB file.
write_pka(protein, parameters[, filename, ...])	Write the pKa-file based on the given protein.
write_scwrl_sequence_file(sequence[, ...])	Write a scwrl sequence file for, e.g., generating a mutated protein

propka.output.**get_bond_order**(*atom1*, *atom2*)

Get the order of a bond between two atoms.

Parameters

- **atom1** – first atom in bond
- **atom2** – second atom in bond

Returns string with bond type

propka.output.**get_charge_profile_section**(*protein*, *conformation='AVR'*, *_=None*)

Returns string with the charge profile section of the results.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **_** – options object

Returns string

propka.output.**get_determinant_section**(*protein*, *conformation*, *parameters*)
Returns string with determinant section of results.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **parameters** – parameters

Returns string

propka.output.**get_determinants_header**()
Create the Determinant header.

Returns string

propka.output.**get_folding_profile_section**(*protein*, *conformation='AVR'*, *direction='folding'*, *reference='neutral'*, *window=[0.0, 14.0, 1.0]*, *_=False*, *__=None*)
Returns string with the folding profile section of the results.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **direction** – 'folding' or other
- **reference** – reference state
- **window** – pH window [min, max, step]
- **_** – Boolean for verbose output
- **__** – options object

Returns string

propka.output.**get_propka_header**()
Create the header.

Returns string

propka.output.**get_references_header**()
Create the 'references' part of output file.

Returns string

propka.output.**get_summary_header**()
Create the summary header.

Returns string

propka.output.**get_summary_section**(*protein*, *conformation*, *parameters*)
Returns string with summary section of the results.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **parameters** – parameters

Returns string

propka.output.**get_the_line**()
Draw the line-Johnny Cash would have been proud-or actually Aerosmith!

NOTE - Johnny Cash walked the line.

Returns string

propka.output.**get_warning_header**()
Create the 'warning' part of the output file.

TODO - this function is essentially a no-op.

Returns string

propka.output.**make_interaction_map**(*name*, *list_*, *interaction*)
Print out an interaction map named 'name' of the groups in 'list' based on the function 'interaction'

Parameters

- **list** – list of groups
- **interaction** – some sort of function

Returns string

propka.output.**open_file_for_writing**(*input_file*)
Open file or file-like stream for writing.

TODO - convert this to a context manager.

Parameters

- **input_file** – path to file or file-like object. If file-like object,
- **will attempt to get file mode.** (*then*) –

propka.output.**print_header**()
Print header section of output.

propka.output.**print_pka_section**(*protein*, *conformation*, *parameters*)
Prints out pKa section of results.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **parameters** – parameters

propka.output.**print_result**(*protein*, *conformation*, *parameters*)
Prints all resulting output from determinants and down.

Parameters

- **protein** – protein object
- **conformation** – specific conformation
- **parameters** – parameters

propka.output.**print_tm_profile**(*protein*, *reference='neutral'*, *window=[0.0, 14.0, 1.0]*, *__=[0.0, 0.0]*, *tms=None*, *ref=None*, *_=False*, *options=None*)
Print Tm profile.

I think Tm refers to the denaturation temperature.

Parameters

- **protein** – protein object
- **reference** – reference state
- **window** – pH window [min, max, step]
- **__** – temperature range [min, max]
- **tms** – TODO - figure this out
- **ref** – TODO - figure this out (probably reference state?)
- **_** – Boolean for verbosity
- **options** – options object

propka.output.**write_file**(*filename*, *lines*)
Writes a new file.

Parameters

- **filename** – name of file
- **lines** – lines to write to file

propka.output.**write_jackal_scap_file**(*mutation_data=None*, *filename='1xxx_scap.list'*, *_=None*)
Write a scap file for, i.e., generating a mutated protein

TODO - figure out what this is

propka.output.**write_mol2_for_atoms**(*atoms*, *filename*)
Write out MOL2 file for atoms.

Parameters

- **atoms** – list of atoms
- **filename** – name of file

propka.output.**write_pdb_for_atoms**(*atoms*, *filename*, *make_conect_section=False*)
Write out PDB file for atoms.

Parameters

- **atoms** – list of atoms
- **filename** – name of file
- **make_conect_section** – generate a CONECT PDB section

propka.output.**write_pdb_for_conformation**(*conformation*, *filename*)
Write PDB conformation to a file.

Parameters

- **conformation** – conformation container
- **filename** – filename for output

propka.output.**write_pdb_for_protein**(*protein*, *pdbfile=None*, *filename=None*, *include_hydrogens=False*, *_=None*)
Write a residue to the new PDB file.

Parameters

- **protein** – protein object

- **pdbfile** – PDB file
- **filename** – file to write to
- **include_hydrogens** – Boolean indicating whether to include hydrogens
- **options** – options object

propka.output.**write_pka**(*protein*, *parameters*, *filename=None*, *conformation='1A'*, *reference='neutral'*, *_='folding'*, *verbose=False*, *__=None*)

Write the pKa-file based on the given protein.

Parameters

- **protein** – protein object
- **filename** – output file name
- **conformation** – TODO - figure this out
- **reference** – reference state
- **_** – "folding" or other
- **verbose** – Boolean flag for verbosity
- **__** – options object

propka.output.**write_scwrl_sequence_file**(*sequence*, *filename='x-ray.seq'*, *_=None*)

Write a scwrl sequence file for, e.g., generating a mutated protein

TODO - figure out what this is

propka.parameters

Configuration file parameters

Holds parameters and settings that can be set in `propka.cfg`. The file format consists of lines of `keyword value [value ...]`, blank lines, and comment lines (introduced with `#`).

The module attributes below list the names and types of all key words in configuration file.

Module Attributes

MATRICES	matrices
PAIR_WISE_MATRICES	pari-wise matrices
NUMBER_DICTIONARIES	`dict` containing numbers
LIST_DICTIONARIES	`dict` containing lists
STRING_DICTIONARIES	`dict` containing strings
STRING_LISTS	`list` containing strings
DISTANCES	distances (`float`)
PARAMETERS	other parameters

Classes

InteractionMatrix(name)	Interaction matrix class.
PairwiseMatrix(name)	Pairwise interaction matrix class.
Parameters()	PROPKA parameter class.

propka.parameters.**DISTANCES = ['desolv_cutoff', 'buried_cutoff', 'coulomb_cutoff1', 'coulo**
distances (float)

class propka.parameters.**InteractionMatrix**(*name*)
Interaction matrix class.

add(*words*)
Add values to matrix.

Parameters **words** – values to add

get_value(*item1*, *item2*)
Get specific matrix value.

Parameters

- **item1** – matrix row index
- **item2** – matrix column index

Returns matrix value or None

keys()
Get keys from matrix.

Returns dictionary key list

propka.parameters.**LIST_DICTIONARIES = ['backbone_NH_hydrogen_bond', 'backbone_CO_hydrogen_**
dict containing lists

propka.parameters.**MATRICES = ['interaction_matrix']**
matrices

propka.parameters.**NUMBER_DICTIONARIES = ['VanDerWaalsVolume', 'charge', 'model_pkas', 'ion**
dict containing numbers

propka.parameters.**PAIR_WISE_MATRICES = ['sidechain_cutoffs']**
pari-wise matrices

propka.parameters.**PARAMETERS = ['Nmin', 'Nmax', 'desolvationSurfaceScalingFactor', 'desolva**
other parameters

class propka.parameters.**PairwiseMatrix**(*name*)
Pairwise interaction matrix class.

add(*words*)
Add information to the matrix.

TODO - this function unnecessarily bundles arguments into a tuple

Parameters **words** – tuple with assignment information and value

get_value(*item1*, *item2*)
Get specified value from matrix.

Parameters

- **item1** – row index

- **item2** – column index

Returns matrix value (or default)

insert(*key1*, *key2*, *value*)
Insert value into matrix.

Parameters

- **key1** – first matrix key (row)
- **key2** – second matrix key (column)
- **value** – value to insert

keys()
Get keys from matrix.

Returns dictionary key list

class propka.parameters.**Parameters**
PROPKA parameter class.

parse_distance(*words*)
Parse field to distance.

Parameters **words** – strings to parse

parse_line(*line*)
Parse parameter file line.

parse_parameter(*words*)
Parse field to parameters.

Parameters **words** – strings to parse

parse_string(*words*)
Parse field to strings.

Parameters **words** – strings to parse

parse_to_list_dictionary(*words*)
Parse field to list dictionary.

Parameters **words** – strings to parse.

parse_to_matrix(*words*)
Parse field to matrix.

Parameters **words** – strings to parse

parse_to_number_dictionary(*words*)
Parse field to number dictionary.

Parameters **words** – strings to parse.

parse_to_string_dictionary(*words*)
Parse field to string dictionary.

Parameters **words** – strings to parse

parse_to_string_list(*words*)
Parse field to string list.

Parameters **words** – strings to parse

print_interaction_parameters()
Print interaction parameters.

print_interaction_parameters_latex()
Print interaction parameters in LaTeX format.

print_interactions_latex()
Print interactions in LaTeX.

set_up_data_structures()
Set up internal data structures.

TODO - it would be better to make these assignments explicit in __init__.

propka.parameters.**STRING_DICTIONARIES = ['protein_group_mapping']**
dict containing strings

propka.parameters.**STRING_LISTS = ['ignore_residues', 'angular_dependent_sidechain_interact**
list containing strings

propka.hybrid36

Hybrid36 PDB-like file format

hybrid36 is an alternative PDB format that can encode larger atom numbers. This module provides the *decode()* functon to parse the atom numbers in hybrid36 "PDB" files.

Functions

decode(input_string)	Convert an input string of a number in hybrid-36 format to an integer.

propka.hybrid36.**decode**(*input_string*)
Convert an input string of a number in hybrid-36 format to an integer.

Parameters **input_string** – input string

Returns integer

propka.ligand_pka_values

Ligand pKa values from Marvin

Ligand pKa values can be obtained from the commercial Marvin software (namely, the **cxcalc** and **molconvert** programs are required).

Classes

LigandPkaValues(parameters)	Ligand pKa value class.

class propka.ligand_pka_values.**LigandPkaValues**(*parameters*)

Ligand pKa value class.

static extract_pkas(*output*)

Extract pKa value from output.

Parameters **output** – output string to parse

Returns

1. Indices
2. Values
3. Types

static find_in_path(*program*)

Find a program in the system path.

Parameters **program** – program to find

Returns location of program

get_marvin_pkas_for_atoms(*atoms*, *name='temp'*, *reuse=False*, *num_pkas=10*, *min_ph=- 10*, *max_ph=20*)

Use Marvin executables to calculate pKas for a list of atoms.

Parameters

- **atoms** – list of atoms
- **name** – filename
- **reuse** – flag to reuse the structure files
- **num_pkas** – number of pKas to calculate
- **min_ph** – minimum pH value
- **max_ph** – maximum pH value

get_marvin_pkas_for_conformation_container(*conformation*, *name='temp'*, *reuse=False*, *num_pkas=10*, *min_ph=-10*, *max_ph=20*)

Use Marvin executables to calculate pKas for a conformation container.

Parameters

- **conformation** – conformation container
- **name** – filename
- **reuse** – flag to reuse the structure files
- **num_pkas** – number of pKas to calculate
- **min_ph** – minimum pH value
- **max_ph** – maximum pH value

get_marvin_pkas_for_molecular_container(*molecule*, *num_pkas=10*, *min_ph=- 10*, *max_ph=20*)

Use Marvin executables to calculate pKas for a molecular container.

Parameters

- **molecule** – molecular container
- **num_pkas** – number of pKas to calculate
- **min_ph** – minimum pH value
- **max_ph** – maximum pH value

get_marvin_pkas_for_molecule(*atoms*, *filename='__tmp_ligand.mol2'*, *reuse=False*, *num_pkas=10*, *min_ph=- 10*, *max_ph=20*)

Use Marvin executables to calculate pKas for a molecule.

Parameters

- **molecule** – the molecule
- **name** – filename
- **reuse** – flag to reuse the structure files
- **num_pkas** – number of pKas to calculate
- **min_ph** – minimum pH value
- **max_ph** – maximum pH value

get_marvin_pkas_for_pdb_file(*molecule*, *parameters*, *num_pkas=10*, *min_ph=- 10*, *max_ph=20*)

Use Marvin executables to get pKas for a PDB file.

Parameters

- **pdbfile** – PDB file
- **molecule** – MolecularContainer object
- **num_pkas** – number of pKas to get
- **min_ph** – minimum pH value
- **max_ph** – maximum pH value

propka.run

Script functionality

The `run` module provides a high-level interface to PROPKA 3.

The **propka3** script consists of the *`main()`* function. If similar functionality is desired from a Python script (without having to call the **propka** script itself) then the *`single()`* function can be used instead.

Functions

main([optargs])	Read in structure files, calculate pKa values, and print pKa files.
single(filename[, optargs, stream, write_pka])	Run a single PROPKA calculation using `filename` as input.

`propka.run.`**`main`**(*optargs=None*)

Read in structure files, calculate pKa values, and print pKa files.

Changed in version 3.4.0: Removed ability to write out PROPKA input files.

`propka.run.`**`single`**(*filename: str*, *optargs: tuple = ()*, *stream=None*, *write_pka: bool = True*)

Run a single PROPKA calculation using `filename` as input.

Parameters

- **`filename`** (*`str`*) – name of input file. If filestream is not passed via `stream`, should be a path to the file to be read.
- **`optargs`** (*`tuple`*) – Optional, commandline options for propka. Extra files passed via `optargs` will be ignored, see Notes.
- **`stream`** – optional filestream handle. If `None`, then `filename` will be used as path to input file for reading.
- **`write_pka`** (*`bool`*) – Controls if the pKa file should be writen to disk.

Returns *`MolecularContainer`* object.

Examples

Given an input file "protein.pdb", run the equivalent of `propka3 --mutation=N25R/N181D -v --pH=7.2 protein.pdb` as:

```
propka.run.single("protein.pdb",
    optargs=["--mutation=N25R/N181D", "-v", "--pH=7.2"])
```

By default, a pKa file will be written. However in some cases one may wish to not output this file and just have access to the *`MolecularContainer`* object. If so, then pass `False` to `write_pka`:

```
mol = propka.run.single("protein.pdb", write_pka=False)
```

In some cases, one may also want to pass a file-like (e.g. `io.StringIO`) object instead of a file path as a string. In these cases the file-like object should be passed to the `stream` argument and a string indicating the file type in the `filename` argument; this string only has to look like a valid file name, it does not need to exist because the data are actually read from `stream`. This approach is necessary because file-like objects do not usually have names, and propka uses the `filename` argument to determine the input file type, and assigns the file name for the *`MolecularContainer`* object:

```
mol = propka.run.single('input.pdb', stream=string_io_file)
```

In this case, a PDB file-like object was passed as *string_io_file*. The resultant pKa file will be written out as *input.pka*.

Notes

- Only a single input structure file will be processed, defined by `filename` (and `stream` if passing a file-like object). Any additional files passed via the *-f* or *–file* flag to optargs will be ignored.

See also:

propka.input.read_molecule_file()

Changed in version 3.4.0: Removed ability to write out PROPKA input files.

propka.version

Version-based configuration

Contains version-specific methods and parameters.

TODO - this module unnecessarily confuses the code. Can we eliminate it?

Classes

ElementBasedLigandInteractions(parameters)	TODO - figure out what this is.
Propka30(parameters)	Version class for PROPKA 3.0.
SimpleHB(parameters)	A simple hydrogen bond version.
Version(parameters)	Store version-specific methods and parameters.
VersionA(parameters)	TODO - figure out what this is.

class propka.version.**ElementBasedLigandInteractions**(*parameters*)
TODO - figure out what this is.

get_backbone_hydrogen_bond_parameters(*backbone_atom*, *atom*)
Get hydrogen bond parameters between backbone atom and other atom.

Parameters

- **backbone_atom** – backbone atom
- **atom** – other atom

Returns [v, [c1, c3]] TODO - figure out what this is

get_hydrogen_bond_parameters(*atom1*, *atom2*)
Get hydrogen bond parameters for two atoms.

Parameters

- **atom1** – first atom
- **atom2** – second atom

Returns [dpka_max, cutoff]

class propka.version.**Propka30**(*parameters*)
Version class for PROPKA 3.0.

get_hydrogen_bond_parameters(*atom1*, *atom2*)
Get hydrogen bond parameters for two atoms.

Parameters

- **atom1** – first atom
- **atom2** – second atom

Returns [dpka_max, cutoff]

class propka.version.**SimpleHB**(*parameters*)

A simple hydrogen bond version.

get_backbone_hydrogen_bond_parameters(*backbone_atom*, *atom*)

Get hydrogen bond parameters between backbone atom and other atom.

Parameters

- **backbone_atom** – backbone atom
- **atom** – other atom

Returns [v, [c1, c3]] TODO - figure out what this is

get_hydrogen_bond_parameters(*atom1*, *atom2*)

Get hydrogen bond parameters for two atoms.

Parameters

- **atom1** – first atom
- **atom2** – second atom

Returns [dpka_max, cutoff]

class propka.version.**Version**(*parameters*)

Store version-specific methods and parameters.

calculate_backbone_reorganization(*conformation*)

Calculate backbone reorganization using assigned model.

calculate_coulomb_energy(*distance*, *weight*)

Calculate Coulomb energy using assigned model.

calculate_desolvation(*group*)

Calculate desolvation energy using assigned model.

calculate_pair_weight(*num_volume1*, *num_volume2*)

Calculate pair weight using assigned model.

calculate_side_chain_energy(*distance*, *dpka_max*, *cutoff*, *_*, *f_angle*)

Calculate sidechain energy using assigned model.

check_coulomb_pair(*group1*, *group2*, *distance*)

Check Coulomb pair using assigned model.

check_exceptions(*group1*, *group2*)

Calculate exceptions using assigned model.

electrostatic_interaction(*group1*, *group2*, *distance*)

Calculate electrostatic energy using assigned model.

static empty_function(**args*)

Placeholder function so we don't use uninitialized variables.

Parameters **args** – whatever arguments would have been passed to the function

Raises **NotImplementedError** –

hydrogen_bond_interaction(*group1*, *group2*)
Calculate H-bond energy using assigned model.

setup_bonding(*molecular_container*)
Setup bonding using assigned model.

setup_bonding_and_protonation(*molecular_container*)
Setup bonding and protonation using assigned model.

class propka.version.**VersionA**(*parameters*)
TODO - figure out what this is.

get_backbone_hydrogen_bond_parameters(*backbone_atom*, *atom*)
Get hydrogen bond parameters between backbone atom and other atom.

Parameters

- **backbone_atom** – backbone atom
- **atom** – other atom

Returns [v, [c1, c3]] TODO - figure out what this is

get_hydrogen_bond_parameters(*atom1*, *atom2*)
Get hydrogen bond parameters for two atoms.

Parameters

- **atom1** – first atom
- **atom2** – second atom

Returns [dpka_max, cutoff]

5.4.3 Structure processing

protonate	Protonate a structure
hydrogens	Hydrogens
ligand	Ligand atom typing

propka.protonate

Protonate a structure

The *Protonate* processes a *propka.molecular_container.MolecularContainer* and adds protons.

Classes

Protonate([verbose])	Protonates atoms using VSEPR theory

class propka.protonate.**Protonate**(*verbose=False*)

Protonates atoms using VSEPR theory

static add_proton(*atom*, *position*)

Add a proton to an atom at a specific position.

Parameters

- **atom** – atom to protonate
- **position** – position for proton

add_protons(*atom*)

Add protons to atom.

Parameters **atom** – atom for calculation

protonate(*molecules*)

Protonate all atoms in the molecular container.

Parameters **molecules** – molecular containers

protonate_atom(*atom*)

Protonate an atom.

Parameters **atom** – atom to be protonated

static remove_all_hydrogen_atoms(*molecular_container*)

Remove all hydrogen atoms from molecule.

Parameters **molecular_container** – molecule to remove hydrogens from

set_bond_distance(*bvec*, *element*)

Set bond distance between atom and element.

Parameters

- **bvec** – bond vector
- **element** – bonded element

Returns scaled bond vector

set_charge(*atom*)

Set charge for atom.

Parameters **atom** – atom to be charged

set_number_of_protons_to_add(*atom*)

Set the number of protons to add to this atom.

Parameters **atom** – atom for calculation

static set_proton_names(*heavy_atoms*)

Set names for protons.

Parameters **heavy_atoms** – list of heavy atoms with protons to be named

set_steric_number_and_lone_pairs(*atom*)

Set steric number and lone pairs for atom.

Parameters **atom** – atom for calculation

tetrahedral(*atom*)
Protonate atom in tetrahedral geometry.

Parameters **atom** – atom to protonate.

trigonal(*atom*)
Add hydrogens in trigonal geometry.

Parameters **atom** – atom to protonate

propka.hydrogens

Hydrogens

Calculations related to hydrogen placement.

Functions

add_amd_hydrogen(residue)	Adds Gln & Asn hydrogen atoms to residues according to the 'old way'.
add_arg_hydrogen(residue)	Adds Arg hydrogen atoms to residues according to the 'old way'.
add_backbone_hydrogen(residue, o_atom, c_atom)	Adds hydrogen backbone atoms to residues according to the old way.
add_his_hydrogen(residue)	Adds His hydrogen atoms to residues according to the 'old way'.
add_trp_hydrogen(residue)	Adds Trp hydrogen atoms to residues according to the 'old way'.
make_new_h(atom, x, y, z)	Add a new hydrogen to an atom at the specified position.
protonate_30_style(molecular_container)	Protonate the molecule.
protonate_average_direction(x1_atom, ...)	Protonates an atom, x1_atom, given a direction.
protonate_direction(x1_atom, x2_atom, x3_atom)	Protonates an atom, x1_atom, given a direction.
protonate_sp2(x1_atom, x2_atom, x3_atom)	Protonates a SP2 atom, given a list of atoms
set_ligand_atom_names(molecular_container)	Set names for ligands in molecular container.
setup_bonding(molecular_container)	Set up bonding for a molecular container.
setup_bonding_and_protonation(...)	Set up bonding and protonation for a molecule.
setup_bonding_and_protonation_30_style(...)	Set up bonding for a molecular container.

propka.hydrogens.**add_amd_hydrogen**(*residue*)
Adds Gln & Asn hydrogen atoms to residues according to the 'old way'.

Parameters **residue** – glutamine or asparagine residue to protonate

propka.hydrogens.**add_arg_hydrogen**(*residue*)
Adds Arg hydrogen atoms to residues according to the 'old way'.

Parameters **residue** – arginine residue to protonate

Returns list of hydrogen atoms

propka.hydrogens.**add_backbone_hydrogen**(*residue*, *o_atom*, *c_atom*)
Adds hydrogen backbone atoms to residues according to the old way.

dR is wrong for the N-terminus (i.e. first residue) but it doesn't affect anything at the moment. Could be improved, but works for now.

Parameters

- **residue** – residue to protonate
- **o_atom** – backbone oxygen atom
- **c_atom** – backbone carbon atom

Returns [new backbone oxygen atom, new backbone carbon atom]

propka.hydrogens.**add_his_hydrogen**(*residue*)
Adds His hydrogen atoms to residues according to the 'old way'.

Parameters **residue** – histidine residue to protonate

propka.hydrogens.**add_trp_hydrogen**(*residue*)
Adds Trp hydrogen atoms to residues according to the 'old way'.

Parameters **residue** – tryptophan residue to protonate

propka.hydrogens.**make_new_h**(*atom*, *x*, *y*, *z*)
Add a new hydrogen to an atom at the specified position.

Parameters

- **atom** – atom to protonate
- **x** – x position of hydrogen
- **y** – y position of hydrogen
- **z** – z position of hydrogen

Returns new hydrogen atom

propka.hydrogens.**protonate_30_style**(*molecular_container*)
Protonate the molecule.

Parameters **molecular_container** – molecule

propka.hydrogens.**protonate_average_direction**(*x1_atom*, *x2_atom*, *x3_atom*)
Protonates an atom, x1_atom, given a direction.

New direction for x1_atom is (x1_atom/x2_atom -> x3_atom). Note, this one uses the average of x1_atom & x2_atom (N & O) unlike the previous N - C = O

Parameters

- **x1_atom** – atom to be protonated
- **x2_atom** – atom for direction
- **x3_atom** – other atom for direction

Returns new hydrogen atom

propka.hydrogens.**protonate_direction**(*x1_atom*, *x2_atom*, *x3_atom*)
Protonates an atom, x1_atom, given a direction.

New direction for x1_atom proton is (x2_atom -> x3_atom).

Parameters

- **x1_atom** – atom to be protonated
- **x2_atom** – atom for direction

- **x3_atom** – other atom for direction

Returns new hydrogen atom

propka.hydrogens.**protonate_sp2**(*x1_atom*, *x2_atom*, *x3_atom*)
Protonates a SP2 atom, given a list of atoms

Parameters

- **x1_atom** – atom to set direction
- **x2_atom** – atom to be protonated
- **x3_atom** – other atom to set direction

Returns new hydrogen atom

propka.hydrogens.**set_ligand_atom_names**(*molecular_container*)
Set names for ligands in molecular container.

Parameters **molecular_container** – molecular container for ligand names

propka.hydrogens.**setup_bonding**(*molecular_container*)
Set up bonding for a molecular container.

Parameters **molecular_container** – the molecular container in question

Returns BondMaker object

propka.hydrogens.**setup_bonding_and_protonation**(*molecular_container*)
Set up bonding and protonation for a molecule.

Parameters

- **parameters** – not used
- **molecular_container** – molecule container.

propka.hydrogens.**setup_bonding_and_protonation_30_style**(*molecular_container*)
Set up bonding for a molecular container.

Parameters **molecular_container** – the molecular container in question

Returns BondMaker object

propka.ligand

Ligand atom typing

This module contains the *assign_sybyl_type()* function to analyze all *propka.atom.Atom* in terms of SYBYL atom types (see *ALL_SYBYL_TYPES*).

Module Attributes

ALL_SYBYL_TYPES	SYBYL atom types
PROPKA_INPUT_TYPES	PROPKA input types

Functions

are_atoms_planar(atoms)	Test whether a group of atoms are planar.
assign_sybyl_type(atom)	Assign Sybyl type to atom.
identify_ring(this_atom, original_atom, ...)	Identify the atoms in a ring
is_aromatic_ring(atoms)	Determine whether group of atoms form aromatic ring.
is_planar(atom)	Finds out if atom forms a plane together with its bonded atoms.
is_ring_member(atom)	Determine if atom is a member of a ring.
set_type(atom, type_)	Set atom type..

propka.ligand.**ALL_SYBYL_TYPES = ['C.3', 'H', 'C.2', 'H.spc', 'C.1', 'H.t3p', 'C.ar', 'LP',**
SYBYL atom types

propka.ligand.**PROPKA_INPUT_TYPES = ['1P', '1N', '2P', '2N', 'C3', 'H', 'C2', 'Hsp', 'C1',**
PROPKA input types

propka.ligand.**are_atoms_planar**(*atoms*)
Test whether a group of atoms are planar.

Parameters **atoms** – list of atoms

Returns Boolean

propka.ligand.**assign_sybyl_type**(*atom*)
Assign Sybyl type to atom.

Parameters **atom** – atom to assign

propka.ligand.**identify_ring**(*this_atom*, *original_atom*, *number*, *past_atoms*)
Identify the atoms in a ring

Parameters

- **this_atom** – atom to test
- **original_atom** – some other atom
- **number** – number of atoms
- **past_atoms** – atoms that have already been found

Returns list of atoms

propka.ligand.**is_aromatic_ring**(*atoms*)
Determine whether group of atoms form aromatic ring.

Parameters **atoms** – list of atoms to test

Returns Boolean

propka.ligand.**is_planar**(*atom*)
Finds out if atom forms a plane together with its bonded atoms.

Parameters **atom** – atom to test

Returns Boolean

propka.ligand.**is_ring_member**(*atom*)
Determine if atom is a member of a ring.

Parameters **atom** – atom to test

Returns list of atoms

propka.ligand.**set_type**(*atom*, *type_*)
Set atom type..

Parameters

- **atom** – atom to set
- **type** – type value to set

5.4.4 Calculations

calculations	Calculations
coupled_groups	Coupling between groups
determinant	Determinant
determinants	Working with Determinants
energy	Energy calculations
iterative	Working with Determinants
vector_algebra	Vector calculations

propka.calculations

Calculations

Mathematical helper functions.

Module Attributes

MAX_DISTANCE	Maximum distance used to bound calculations of smallest distance

Functions

distance(atom1, atom2)	Calculate the distance between two atoms.
get_smallest_distance(atoms1, atoms2)	Calculate the smallest distance between two groups of atoms.
squared_distance(atom1, atom2)	Calculate the squared distance between two atoms.

propka.calculations.**MAX_DISTANCE = 1000000.0**
Maximum distance used to bound calculations of smallest distance

propka.calculations.**distance**(*atom1*, *atom2*)
Calculate the distance between two atoms.

Parameters

- **atom1** – first atom for distance calculation
- **atom2** – second atom for distance calculation

Returns distance

propka.calculations.**get_smallest_distance**(*atoms1*, *atoms2*)
Calculate the smallest distance between two groups of atoms.

Parameters

- **atoms1** – atom group 1
- **atoms2** – atom group 2

Returns smallest distance between groups

propka.calculations.**squared_distance**(*atom1*, *atom2*)
Calculate the squared distance between two atoms.

Parameters

- **atom1** – first atom for distance calculation
- **atom2** – second atom for distance calculation

Returns distance squared

propka.coupled_groups

Coupling between groups

Describe and analyze energetic coupling between groups.

Classes

NonCovalentlyCoupledGroups()	Groups that are coupled without covalent bonding.

class propka.coupled_groups.**NonCovalentlyCoupledGroups**
Groups that are coupled without covalent bonding.

get_free_energy_diff_factor(*energy1*, *energy2*)
Get scaling factor for difference between free energies.

Parameters

- **energy1** – first energy to compare
- **energy2** – second energy to compare

Returns float value of scaling factor

static get_interaction(*group1*, *group2*, *include_side_chain_hbs=True*)
Get interaction energy between two groups.

Parameters

- **group1** – first group for interaction
- **group2** – second group for interaction

- **include_side_chain_hbs** – include sidechain hydrogen bonds in energy

Returns interaction energy (float)

get_interaction_factor(*interaction_energy*)
Get scaling factor related to interaction energy.

Parameters **interaction_energy** – interaction energy

Returns float value of scaling factor

get_pka_diff_factor(*pka1*, *pka2*)
Get scaling factor for difference between intrinsic pKa values.

Parameters

- **pka1** – first pKa to compare
- **pka2** – second pKa to compare

Returns float value of scaling factor

identify_non_covalently_coupled_groups(*conformation*, *verbose=True*)
Find coupled residues in protein.

Parameters

- **conformation** – protein conformation to test
- **verbose** – verbose output (boolean)

is_coupled_protonation_state_probability(*group1*, *group2*, *energy_method*, *return_on_fail=True*)
Check whether two groups are energetically coupled.

Parameters

- **group1** – first group for interaction
- **group2** – second group for interaction
- **energy_method** – function for calculating energy
- **return_on_fail** – return if part of the calculation fails

Returns dictionary describing coupling

static make_data_to_string(*data*, *group1*, *group2*)
Describe interaction between groups.

Parameters

- **data** – data about interactions
- **group1** – first group
- **group2** – second group

Returns formatted string with information.

print_determinants_section(*system*, *tag*)
Print determinants of system.

Parameters

- **system** – set of groups
- **tag** – something to add to output

Returns string with summary

print_out_swaps(*conformation*)
Print out something having to do with coupling interactions.

Parameters **conformation** – conformation to print

print_system(*conformation*, *system*)
Print out something about the system.

Parameters

- **conformation** – conformation to print
- **system** – system to print

swap_interactions(*groups1*, *groups2*, *include_side_chain_hbs=True*)
Swap interactions between two groups.

Parameters

- **group1** – first group to swap
- **group2** – second group to swap

static tagged_format(*tag*, *str_*, *labels*)
Tag a string.

Parameters

- **tag** – tag to add
- **str** – string to tag
- **labels** – labels to replace

Returns tagged string

static transfer_determinant(*determinants1*, *determinants2*, *label1*, *label2*)
Transfer information between two sets of determinants.

Parameters

- **determinants1** – determinant list
- **determinants2** – determinant list
- **label1** – label for list 1
- **label2** – label for list 2

propka.determinant

Determinant

Provides the *Determinant* class.

See also:

- *propka.determinants*
- *propka.iterative*

Classes

Determinant(group, value)	Determinant class.

class propka.determinant.**Determinant**(*group*, *value*)

Determinant class.

Appears to be a container for storing information and values about groups that interact to influence titration states.

TODO - figure out what this class does.

add(*value*)

Increment determinant value.

Parameters **value** – value to add to determinant

propka.determinants

Working with Determinants

Functions to manipulate *propka.determinant.Determinant* objects.

See also:

propka.determinant

Functions

add_coulomb_acid_pair(object1, object2, value)	Add the Coulomb interaction (an acid pair).
add_coulomb_base_pair(object1, object2, value)	Add the Coulomb interaction (a base pair).
add_coulomb_determinants(group1, group2, ...)	Add non-iterative Coulomb determinants and perturbations.
add_coulomb_ion_pair(object1, object2, value)	Add the Coulomb interaction (an acid-base pair).
add_determinants(group1, group2, distance, ...)	Add determinants and perturbations for distance(R1,R2) < coulomb_cutoff.
add_sidechain_determinants(group1, group2[, ...])	Add side-chain determinants and perturbations.
set_backbone_determinants(titratable_groups, ...)	Set determinants between titrable and backbone groups.
set_determinants(propka_groups[, version, ...])	Add side-chain and coulomb determinants/perturbations to all residues.
set_ion_determinants(conformation_container, ...)	Add ion determinants and perturbations.

propka.determinants.**add_coulomb_acid_pair**(*object1*, *object2*, *value*)

Add the Coulomb interaction (an acid pair).

The higher pKa is raised.

Parameters

- **object1** – first part of pair
- **object2** – second part of pair
- **value** – determinant value

propka.determinants.**add_coulomb_base_pair**(*object1*, *object2*, *value*)
Add the Coulomb interaction (a base pair).

The lower pKa is lowered.

Parameters

- **object1** – first part of pair
- **object2** – second part of pair
- **value** – determinant value

propka.determinants.**add_coulomb_determinants**(*group1*, *group2*, *distance*, *version*)
Add non-iterative Coulomb determinants and perturbations.

Parameters

- **group1** – first group to add
- **group2** – second group to add
- **distance** – distance between groups
- **version** – version object

propka.determinants.**add_coulomb_ion_pair**(*object1*, *object2*, *value*)
Add the Coulomb interaction (an acid-base pair).

The pKa of the acid is lowered & the pKa of the base is raised.

Parameters

- **object1** – first part of pair
- **object2** – second part of pair
- **value** – determinant value

propka.determinants.**add_determinants**(*group1*, *group2*, *distance*, *version*)
Add determinants and perturbations for distance(R1,R2) < coulomb_cutoff.

Parameters

- **group1** – first group to add
- **group2** – second group to add
- **distance** – distance between groups
- **version** – version object

propka.determinants.**add_sidechain_determinants**(*group1*, *group2*, *version=None*)
Add side-chain determinants and perturbations.

NOTE - res_num1 > res_num2

Parameters

- **group1** – first group to add
- **group2** – second group to add
- **version** – version object

propka.determinants.**set_backbone_determinants**(*titratable_groups*, *backbone_groups*, *version*)

Set determinants between titrable and backbone groups.

Parameters

- **titratable_groups** – list of titratable groups
- **backbone_groups** – list of backbone groups
- **version** – version object

propka.determinants.**set_determinants**(*propka_groups*, *version=None*, *options=None*)

Add side-chain and coulomb determinants/perturbations to all residues.

NOTE - backbone determinants are set separately

Parameters

- **propka_groups** – groups to adjust
- **version** – version object
- **options** – options object

propka.determinants.**set_ion_determinants**(*conformation_container*, *version*)

Add ion determinants and perturbations.

Parameters

- **conformation_container** – conformation to set
- **version** – version object

propka.energy

Energy calculations

Energy calculations.

Functions

angle_distance_factors([atom1, atom2, ...])	Calculate distance and angle factors for three atoms for backbone interactions.
backbone_reorganization(_, conformation)	Perform calculations related to backbone reorganizations.
calculate_pair_weight(parameters, ...)	Calculate the atom-pair based desolvation weight.
calculate_scale_factor(parameters, weight)	Calculate desolvation scaling factor.
calculate_weight(parameters, num_volume)	Calculate the atom-based desolvation weight.
check_buried(num_volume1, num_volume2)	Check to see if an interaction is buried
check_coo_arg_exception(group_coo, ...)	Check for COO-ARG interaction atypical behavior.
check_coo_coo_exception(group1, group2, version)	Check for COO-COO hydrogen-bond atypical interaction behavior.
check_coo_his_exception(group1, group2, version)	Check for COO-HIS atypical interaction behavior
check_coulomb_pair(parameters, group1, ...)	Checks if this Coulomb interaction should be done.

continues on next page

Table 31 – continued from previous page

check_cys_cys_exception(group1, group2, version)	Check for CYS-CYS atypical interaction behavior
check_cys_his_exception(group1, group2, version)	Check for CYS-HIS atypical interaction behavior
check_exceptions(version, group1, group2)	Checks for atypical behavior in interactions between two groups.
check_oco_his_exception(group1, group2, version)	Check for OCO-HIS atypical interaction behavior
coulomb_energy(dist, weight, parameters)	Calculates the Coulomb interaction pKa shift based on Coulomb's law.
electrostatic_interaction(group1, group2, ...)	Calculate electrostatic interaction betwee two groups.
hydrogen_bond_energy(dist, dpka_max, cutoffs)	Calculate hydrogen-bond interaction pKa shift.
hydrogen_bond_interaction(group1, group2, ...)	Calculate energy for hydrogen bond interactions between two groups.
radial_volume_desolvation(parameters, group)	Calculate desolvation terms for group.

propka.energy.**angle_distance_factors**(*atom1=None*, *atom2=None*, *atom3=None*, *center=None*)

Calculate distance and angle factors for three atoms for backbone interactions.

NOTE - you need to use atom1 to be the e.g. ASP atom if distance is reset at return: [O1 – H2-N3].

Also generalized to be able to be used for residue 'centers' for C=O COO interactions.

Parameters

- **atom1** – first atom for calculation (could be None)
- **atom2** – second atom for calculation
- **atom3** – third atom for calculation
- **center** – center point between atoms 1 and 2

Returns

[distance factor between atoms 1 and 2, angle factor, distance factor between atoms 2 and 3]

propka.energy.**backbone_reorganization**(*_*, *conformation*)

Perform calculations related to backbone reorganizations.

NOTE - this was described in the code as "adding test stuff" NOTE - this function does not appear to be used TODO - figure out why a similar function exists in version.py

Parameters

- *_* – not used
- **conformation** – specific molecule conformation

propka.energy.**calculate_pair_weight**(*parameters*, *num_volume1*, *num_volume2*)

Calculate the atom-pair based desolvation weight.

Parameters

- **num_volume1** – number of heavy atoms within first desolvation volume
- **num_volume2** – number of heavy atoms within second desolvation volume

Returns desolvation weight

propka.energy.**calculate_scale_factor**(*parameters*, *weight*)
Calculate desolvation scaling factor.

Parameters

- **parameters** – parameters for desolvation calculation
- **weight** – weight for scaling factor

Returns scaling factor

propka.energy.**calculate_weight**(*parameters*, *num_volume*)
Calculate the atom-based desolvation weight.

TODO - figure out why a similar function exists in version.py

Parameters

- **parameters** – parameters for desolvation calculation
- **num_volume** – number of heavy atoms within desolvation calculation volume

Returns desolvation weight

propka.energy.**check_buried**(*num_volume1*, *num_volume2*)
Check to see if an interaction is buried

Parameters

- **num_volume1** – number of buried heavy atoms in volume 1
- **num_volume2** – number of buried heavy atoms in volume 2

Returns True if interaction is buried, False otherwise

propka.energy.**check_coo_arg_exception**(*group_coo*, *group_arg*, *version*)
Check for COO-ARG interaction atypical behavior.

Uses the two shortest unique distances (involving 2+2 atoms)

Parameters

- **group_coo** – COO group
- **group_arg** – ARG group
- **version** – version object

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_coo_coo_exception**(*group1*, *group2*, *version*)
Check for COO-COO hydrogen-bond atypical interaction behavior.

Parameters

- **group1** – first group for check
- **group2** – second group for check
- **version** – version object

Returns

1. Boolean indicating atypical behavior,

2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_coo_his_exception**(*group1*, *group2*, *version*)
Check for COO-HIS atypical interaction behavior

Parameters

- **group1** – first group for check
- **group2** – second group for check
- **version** – version object

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_coulomb_pair**(*parameters*, *group1*, *group2*, *dist*)
Checks if this Coulomb interaction should be done.

NOTE - this is a propka2.0 hack TODO - figure out why a similar function exists in version.py

Parameters

- **parameters** – parameters for Coulomb calculations
- **group1** – first interacting group
- **group2** – second interacting group
- **dist** – distance between groups

Returns Boolean

propka.energy.**check_cys_cys_exception**(*group1*, *group2*, *version*)
Check for CYS-CYS atypical interaction behavior

Parameters

- **group1** – first group for check
- **group2** – second group for check
- **version** – version object

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_cys_his_exception**(*group1*, *group2*, *version*)
Check for CYS-HIS atypical interaction behavior

Parameters

- **group1** – first group for check
- **group2** – second group for check
- **version** – version object

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_exceptions**(*version*, *group1*, *group2*)
Checks for atypical behavior in interactions between two groups. Checks are made based on group type.

TODO - figure out why a similar function exists in version.py

Parameters

- **version** – version object
- **group1** – first group for check
- **group2** – second group for check

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**check_oco_his_exception**(*group1*, *group2*, *version*)
Check for OCO-HIS atypical interaction behavior

Parameters

- **group1** – first group for check
- **group2** – second group for check
- **version** – version object

Returns

1. Boolean indicating atypical behavior,
2. value associated with atypical interaction (None if Boolean is False)

propka.energy.**coulomb_energy**(*dist*, *weight*, *parameters*)
Calculates the Coulomb interaction pKa shift based on Coulomb's law.

Parameters

- **dist** – distance for electrostatic interaction
- **weight** – scaling of dielectric constant
- **parameters** – parameter object for calculation

Returns pKa shift

propka.energy.**electrostatic_interaction**(*group1*, *group2*, *dist*, *version*)
Calculate electrostatic interaction betwee two groups.

Parameters

- **group1** – first interacting group
- **group2** – second interacting group
- **dist** – distance between groups
- **version** – version-specific object with parameters and functions

Returns electrostatic interaction energy or None (if no interaction is appropriate)

propka.energy.**hydrogen_bond_energy**(*dist*, *dpka_max*, *cutoffs*, *f_angle=1.0*)
Calculate hydrogen-bond interaction pKa shift.

Parameters

- **dist** – distance for hydrogen bond

- **dpka_max** – maximum pKa value shift
- **cutoffs** – array with max and min distance values
- **f_angle** – angle scaling factor

Returns pKa shift value

propka.energy.**hydrogen_bond_interaction**(*group1*, *group2*, *version*)
Calculate energy for hydrogen bond interactions between two groups.

Parameters

- **group1** – first interacting group
- **group2** – second interacting group
- **version** – an object that contains version-specific parameters

Returns hydrogen bond interaction energy

propka.energy.**radial_volume_desolvation**(*parameters*, *group*)
Calculate desolvation terms for group.

Parameters

- **parameters** – parameters for desolvation calculation
- **group** – group of atoms for calculation

propka.iterative

Working with Determinants

Iterative functions for pKa calculations. These appear to mostly involve *propka.determinant.Determinant* instances.

Functions

add_determinants(iterative_interactions, version)	Add determinants iteratively.
add_iterative_acid_pair(object1, object2, ...)	Add the Coulomb 'iterative' interaction (an acid pair).
add_iterative_base_pair(object1, object2, ...)	Add the Coulomb 'iterative' interaction (a base pair).
add_iterative_ion_pair(object1, object2, ...)	Add the Coulomb 'iterative' interaction (an acid-base pair)
add_to_determinant_list(group1, group2, ...)	Add iterative determinantes to the list.
find_iterative(pair, iteratives)	Find the 'iteratives' that correspond to the groups in 'pair'.

Classes

Iterative(group)	Iterative class - pKa values and references of iterative groups.

class propka.iterative.**Iterative**(*group*)
Iterative class - pKa values and references of iterative groups.

NOTE - this class has a fake determinant list, true determinants are made after the iterations are finished.

propka.iterative.**add_determinants**(*iterative_interactions*, *version*, *_=None*)
Add determinants iteratively.

The iterative pKa scheme. Later it is all added in 'calculateTotalPKA'

Parameters

- **iterative_interactions** – list of iterative interactions
- **version** – version object
- **_** – options object

propka.iterative.**add_iterative_acid_pair**(*object1*, *object2*, *interaction*)
Add the Coulomb 'iterative' interaction (an acid pair).

The higher pKa is raised with QQ+HB The lower pKa is lowered with HB

Parameters

- **object1** – first object in pair
- **object2** – second object in pair
- **interaction** – list with [values, annihilation]

propka.iterative.**add_iterative_base_pair**(*object1*, *object2*, *interaction*)
Add the Coulomb 'iterative' interaction (a base pair).

The lower pKa is lowered

Parameters

- **object1** – first object in pair
- **object2** – second object in pair
- **interaction** – list with [values, annihilation]

propka.iterative.**add_iterative_ion_pair**(*object1*, *object2*, *interaction*, *version*)
Add the Coulomb 'iterative' interaction (an acid-base pair)

the pKa of the acid is lowered & the pKa of the base is raised

Parameters

- **object1** – first object in pair
- **object2** – second object in pair
- **interaction** – list with [values, annihilation]
- **version** – version object

propka.iterative.**add_to_determinant_list**(*group1*, *group2*, *distance*, *iterative_interactions*, *version*)

Add iterative determinantes to the list.

[[R1, R2], [side-chain, coulomb], [A1, A2]], ...

NOTE - sign is determined when the interaction is added to the iterative object!

NOTE - distance < coulomb_cutoff here

Parameters

- **group1** – first group in pair
- **group2** – second group in pair
- **distance** – distance between groups
- **iterative_interactions** – interaction list to modify
- **version** – version object

propka.iterative.**find_iterative**(*pair*, *iteratives*)

Find the 'iteratives' that correspond to the groups in 'pair'.

Parameters

- **pair** – groups to match
- **iteratives** – list of iteratives to search

Returns

1. first matched iterative
2. second matched iterative

propka.vector_algebra

Vector calculations

Vector algebra for PROPKA.

Functions

angle(avec, bvec)	Get the angle between two vectors.
angle_degrees(avec, bvec)	Get the angle between two vectors in degrees.
rotate_atoms_around_y_axis(theta)	Get rotation matrix for y-axis.
rotate_atoms_around_z_axis(theta)	Get rotation matrix for z-axis.
rotate_multi_vector_around_an_axis(theta, ...)	Rotate a multi-vector around an axis.
rotate_vector_around_an_axis(theta, axis, vec)	Rotate vector around an axis.
signed_angle_around_axis(avec, bvec, axis)	Get signed angle of two vectors around axis in radians.

Classes

Matrix4x4([a11i, a12i, a13i, a14i, a21i, ...])	A 4-by-4 matrix class.
MultiVector([atom1, atom2])	Collection of vectors for multiple configurations of atoms.
Vector([xi, yi, zi, atom1, atom2])	

class propka.vector_algebra.**Matrix4x4**(*a11i=0.0*, *a12i=0.0*, *a13i=0.0*, *a14i=0.0*, *a21i=0.0*, *a22i=0.0*, *a23i=0.0*, *a24i=0.0*, *a31i=0.0*, *a32i=0.0*, *a33i=0.0*, *a34i=0.0*, *a41i=0.0*, *a42i=0.0*, *a43i=0.0*, *a44i=0.0*)

A 4-by-4 matrix class.

class propka.vector_algebra.**MultiVector**(*atom1=None*, *atom2=None*)

Collection of vectors for multiple configurations of atoms.

TODO - this class does not appear to be used or covered by tests

do_job(*job*)

Append vectors to configuration.

Parameters **job** – name of function to apply to vectors

Returns TODO - figure out what this is

generic_operation(*operation*, *other*)

Perform a generic operation between two MultiVector objects.

Parameters

- **operation** – operation to perform (string)
- **other** – other MultiVector object

static generic_self_operation(*_*)

TODO - delete this.

property get_result

Return the latest result.

rescale(*new_length*)

Rescale multi-vector to new length.

Parameters **new_length** – new length for multi-vector

Result: MultiVector object

class propka.vector_algebra.**Vector**(*xi=0.0*, *yi=0.0*, *zi=0.0*, *atom1=None*, *atom2=None*)

length()

Return vector length.

orthogonal()

Returns a vector orthogonal to self

rescale(*new_length*)

Rescale vector to new length while preserving direction

sq_length()

Return vector squared-length

propka.vector_algebra.**angle**(*avec*, *bvec*)
Get the angle between two vectors.

Parameters

- **avec** – vector 1
- **bvec** – vector 2

Returns angle in radians

propka.vector_algebra.**angle_degrees**(*avec*, *bvec*)
Get the angle between two vectors in degrees.

Parameters

- **avec** – vector 1
- **bvec** – vector 2

Returns angle in degrees

propka.vector_algebra.**rotate_atoms_around_y_axis**(*theta*)
Get rotation matrix for y-axis.

Parameters **theta** – angle of rotation (radians)

Returns rotation matrix

propka.vector_algebra.**rotate_atoms_around_z_axis**(*theta*)
Get rotation matrix for z-axis.

Parameters **theta** – angle of rotation (radians)

Returns rotation matrix

propka.vector_algebra.**rotate_multi_vector_around_an_axis**(*theta*, *axis*, *vec*)
Rotate a multi-vector around an axis.

NOTE - both axis ans v must be MultiVectors.

Parameters

- **theta** – angle (in radians)
- **axis** – multi-vector axis
- **vec** – multi-vector vector

propka.vector_algebra.**rotate_vector_around_an_axis**(*theta*, *axis*, *vec*)
Rotate vector around an axis.

Parameters

- **theta** – rotation angle (in radians)
- **axis** – axis for rotation
- **vec** – vector to rotate

Returns rotated vector

propka.vector_algebra.**signed_angle_around_axis**(*avec*, *bvec*, *axis*)
Get signed angle of two vectors around axis in radians.

Parameters

- **avec** – vector 1

- **bvec** – vector 2
- **axis** – axis

Returns angle in radians

5.5 Changelog

5.5.1 v3.4.0

Changes

- Removed PROPKA input support and argument `--generate-propka-input` (#99)
- Add Python 3.9 support to continuous integration. (#101)
- Removed logging abstraction from code to facilitate debugging and reduce code bloat. (#108)

Fixes

- Fixed bug that raised exception when missing amide nitrogen or oxygen. (#17)
- `propka --version` now shows the program version and exits. Previously this option took a version argument to specify the sub-version of propka. However, this was non-functional at least since 2012. (#89)
- Fix pI reporting in last line of `.pka` file. (https://github.com/jensengroup/propka/pull/91)
- Report correct version in `.pka` file header. (https://github.com/jensengroup/propka/pull/92)
- Fix handling of multi-model PDB without MODEL 1 entry. (https://github.com/jensengroup/propka/issues/96)
- Fixed bug and sped up algorithm for identifying bonds via bounding boxes. (#97, #110)
- Fixed bug in `propka --display-coupled-residues` that crashed the program. (#105)

5.5.2 v3.3.0

Additions

- Add Sphinx documentation on readthedocs.io (#69, #76, #79)

Changes

- Updated `read_molecule_file()` to accept file-like objects. (#83)
- Use versioneer for version management. (#87)
- Add code coverage to continuous integration pipeline. (#62, #71, #76)

Fixes

- Bundle required JSON files with package. (#48)
- Fixed `KeyError` bug in `read_parameter_file()`. (#65)
- Update links to web server. (#80)
- Fixed PDB reading for PROPKA "single" runs. (#82)

5.5.3 v3.2.0

Additions

- Significantly expanded testing framework. (#30, #36, #37)

Changes

- Improved ability to use PROPKA as a module in other Python scripts. (#8)
- Improved output via `logging`. (#11, #12)
- Replaced data/parameter pickle file with human-readable JSON. (#29)
- Significant delinting and formatting standardization against PEP8. (#33, #40)
- Improved package documentation. (#41, #61)
- Significant package refactoring. (#46, #47, #59)
- Simplify module import structure. (#49, #61)
- Improved tempfile handling. (#61)

5.5.4 v3.1.0

Archaeologists wanted to help us document the history of the code in versions 3.1.0 and earlier.

5.6 References

BIBLIOGRAPHY

[Sondergaard2011] C. R. Søndergaard, M. H. M. Olsson, M. Rostkowski, and J. H. Jensen. Improved treatment of ligands and coupling effects in empirical calculation and rationalization of pKa values. *Journal of Chemical Theory and Computation*, 7(7):2284–2295, 2011. doi: 10.1021/ct200133y

[Olsson2011] M. H. M. Olsson, C. R. Søndergaard, M. Rostkowski, and J. H. Jensen. PROPKA3: Consistent treatment of internal and surface residues in empirical pKa predictions. *Journal of Chemical Theory and Computation*, 7(2):525–537, 2011. doi: 10.1021/ct100578z

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W