pdb2pqr

Dec 22, 2020

Contents

CHAPTER 1

Overview

The use of continuum solvation methods such as APBS requires accurate and complete structural data as well as force field parameters such as atomic charges and radii. Unfortunately, the limiting step in continuum electrostatics calculations is often the addition of missing atomic coordinates to molecular structures from the Protein Data Bank and the assignment of parameters to these structures. To address this problem, we have developed PDB2PQR. This software automates many of the common tasks of preparing structures for continuum solvation calculations as well as many other types of biomolecular structure modeling, analysis, and simulation. These tasks include:

- Adding a limited number of missing heavy (non-hydrogen) atoms to biomolecular structures.
- Estimating titration states and protonating biomolecules in a manner consistent with favorable hydrogen bonding.
- Assigning charge and radius parameters from a variety of force fields.
- Generating "PQR" output compatible with several popular computational modeling and analysis packages.
- This service is intended to facilitate the setup and execution of electrostatics calculations for both experts and non-experts and thereby broaden the accessibility of biomolecular solvation and electrostatics analyses to the biomedical community.

CHAPTER 2

Contents

2.1 Getting PDB2PQR

Note: *Before you begin!* PDB2PQR funding is dependent on your help for continued development and support. Please register before using the software so we can accurately report the number of users to our funding agencies.

2.1.1 Web servers

Most functionality is available through our online web servers.

The PDB2PQR web server offers a simple way to use both APBS and PDB2PQR without the need to download and install additional programs.

After registering, please visit http://server.poissonboltzmann.org/ to access the web server.

2.1.2 Python Package Installer (PIP)

Most users who want to use the software offline should install it via **pip** with the following command

```
pip install pdb2pqr
```

from within your favorite virtual environment.

The PIP package provides the *pdb2pqr* Python module as well as the **pdb2pqr30** program which can be used from the command line.

2.1.3 Installation from source code

You can also download the source code from GitHub (we recommend using a tagged release) and building the code yourself with:

```
pip install .
```

from the top-level of the source code directory. Note that developers may want to install the code in "editable" mode with

```
pip install -e .
```

Testing

The software can be tested for correct functioning via Coverage.py with

or pytest

from the top level of the source directory.

2.2 Using PDB2PQR

Note: *Before you begin!* PDB2PQR funding is dependent on your help for continued development and support. Please register before using the software so we can accurately report the number of users to our funding agencies.

PDB2PQR is often used together with the APBS software; e.g., ,in the following type of workflow

1. Start with a PDB ID or locally generated PDB file (see *PDB molecular structure format*).
2. Assign titration states and parameters with **`pdb2pqr`** to convert the biomolecule and ligands to PQR format (see *PQR molecular structure format*).
3. Perform electrostatics calculations with **`apbs`** (can be done from within the PDB2PQR web server).
4. Visualize results from within PDB2PQR web server or with *Other software*.

2.2.1 Web server use

Most users will use PDB2PQR through the web server (after registering, of course). However, it is also possible to install local versions of PDB2PQR and run these through the command line.

2.2.2 Command line use

```
pdb2pqr30 [options] --ff={forcefield} {path} {output-path}
```

This module takes a PDB file as input and performs optimizations before yielding a new PQR-style file in `{output-path}`. If `{path}` is a PDB ID it will automatically be retrieved from the online PDB archive.

In addition to the required `{path}` and `{output-path}` arguments, **`pdb2pqr30`** requires one of the following options:

- `--ff=FIELD_NAME` specifying the forcefield to use. Run `pdb2pqr30 --help` to see specific options.
- `--userff=USER_FIELD_FILE` specifying a user-created forcefield file. Requires `--usernames` and overrides `--ff`.
- `--clean` specifying no optimization, atom addition, or parameter assignment, just return the original PDB file in aligned format. Overrides `--ff` and `--userff` options.

Information about additional options can be obtained by running:

```
pdb2pqr30 --help
```

Additional command-line tools

The following tools are also provided with PDB2PQR. They all accept the `--help` option which provides information about usage of these tools.

dx2cube

Converts an OpenDX volumetric (e.g., as generated by APBS) to a Gaussian cube-format file.

inputgen

Generates an APBS input file with recommended settings.

psize

Provides size information about the molecular system and suggests APBS settings.

2.2.3 Examples and algorithms

Examples

In order to perform electrostatics calculations on your biomolecular structure of interest, you need to provide atomic charge and radius information to APBS. Charges are used to form the biomolecular charge distribution for the Poisson-Boltzmann (PB) equation while the radii are used to construct the dielectric and ionic accessibility functions.

The PDB2PQR web service and software will convert most PDB files into PQR format with some caveats. Although PDB2PQR can fix some missing heavy atoms in sidechains, it does not currently have the (nontrivial) capability to model in large regions of missing backbone and sidechain coordinates. Be patient and make certain that the job you submitted to the PDB2PQR website has finished and you have downloaded the resulting PQR file correctly. It usually takes less than 10 minutes for the job to finish.

These examples assume that you have registered and have access to the PDB2PQR web server.

Adding hydrogens and assigning parameters with the PDB2PQR web server

This example uses http://server.poissonboltzmann.org.

Pick a structure

Start by choosing a PDB file to process. Either enter the 4-character PDB ID into PDB2PQR or accession number (1FAS is a good starting choice) or upload your own PDB file. Note that, if you choose to enter a 4-character PDB ID, PDB2PQR will process all recognizable chains of PDB file as it was deposited in the PDB (e.g., not the biological unit, any related transformations, etc.).

Pick a forcefield

For most applications, the choice is easy: PARSE. This forcefield has been optimized for implicit solvent calculation and is probably the best choice for visualization of biomolecular electrostatics and many common types of energetic calculations for biomolecules. However, AMBER and CHARMM may be more appropriate if you are attempting to compare directly to simulations performed with those force fields, require nucleic acid support, are simulating ligands parameterized with those force fields, etc.

It is also possible to upload a user-defined forcefield (e.g., to define radii and charges for ligands or unusual residues). Please see *Extending PDB2PQR* for more information.

Pick a naming scheme

This choice is largely irrelevant to electrostatics calculations but may be important for some visualization programs. When in doubt, choose the "Internal naming scheme" which attempts to conform to IUPAC standards.

Reconstruct missing atoms (hydrogens)

Under options, be sure the "Ensure that new atoms are not rebuilt too close to existing atoms" and "Optimize the hydrogen bonding network" options are selected. You can select other options as well, if interested.

Download and view the results

Download the resulting PQR file and visualize in a molecular graphics program to examine how the hydrogens were added and how hydrogen bonds were optimized.

Parameterizing ligands with the PDB2PQR web server

This section outlines the parameterization of ligands using the PEOE_PB methods (see Czodrowski et al, 2006 for more information).

The PDB structure 1HPX includes HIV-1 protease complexed with an inhibitor at 2.0 Å resolution. HIV-1 protease has two chains; residue D25 is anionic on one chain and neutral on the other – these titration states are important in the role of D25 as an acid in the catalytic mechanism.

Ignoring the ligand

If we don't want to include the ligand, then the process is straightforward:

1. From the PDB2PQR server web page, enter `1HPX` into the PDB ID field.
2. Choose whichever forcefield and naming schemes you prefer.
3. Under options, be sure the "Ensure that new atoms are not rebuilt too close to existing atoms", "Optimize the hydrogen bonding network", and "Use PROPKA to assign protonation states at pH" options are selected. Choose pH 7 for your initial calculations. You can select other options as well, if interested.
4. Hit the "Submit" button.
5. Once the calculations are complete, you should see a web page with a link to the PROPKA output, a new PQR file, and warnings about the ligand KNI (since we didn't choose to parameterize it in this calculation). For

comparison, you might download the the original PDB file and compare the PDB2PQR-generated structure with the original to see where hydrogens were placed.

Parameterizing the ligand

This section outlines the parameterization of ligands using the PEOE_PB methods (see DOI:10.1002/prot.21110).

Ligand parameterization currently requires a *MOL2-format* representation of the ligand to provide the necessary bonding information. MOL2-format files can be obtained through the PRODRG web server or some molecular modeling software packages. PRODRG provides documentation as well as several examples on ligand preparation on its web page.

We're now ready to look at the 1HPV crystal structure from above and parameterize its ligand, KNI-272.

1. From the PDB2PQR server web page, enter `1HPX` into the PDB ID field.
2. Choose whichever forcefield and naming schemes you prefer.
3. Under options, be sure the "Ensure that new atoms are not rebuilt too close to existing atoms", "Optimize the hydrogen bonding network", and "Assign charges to the ligand specified in a MOL2 file" options are selected. You can select other options as well, if interested.
4. Hit the "Submit" button.
5. Once the calculations are complete, you should see a web page with a link to the new PQR file with a warning about debumping P81 (but no warnings about ligand parameterization!).

As a second example, we use the PDB structure 1ABF of L-arabinose binding protein in complex with a sugar ligand at 1.90 Å resolution. To parameterize both this protein and its ligand:

1. From the PDB2PQR server web page, enter *1ABF* into the PDB ID field.
2. Choose whichever forcefield and naming schemes you prefer.
3. Under options, be sure the "Ensure that new atoms are not rebuilt too close to existing atoms", "Optimize the hydrogen bonding network", and "Assign charges to the ligand specified in a MOL2 file" options are selected. You can select other options as well, if interested.
4. Hit the "Submit" button.
5. Once the calculations are complete, you should see a web page with a link to the new PQR file with a warning about debumping P66, K295, and K306 (but no warnings about ligand parameterization!).

Algorithms used by PDB2PQR

Debumping

Unless otherwise instructed with `--nodebump`, PDB2PQR will attempt to remove steric clashes (debump) between residues.

To determine if a residue needs to be debumped, PDB2PQR compares its atoms to all nearby atoms. With the exception of donor/acceptor pairs and CYS residue SS bonded pairs, a residue needs to be debumped if any of its atoms are within cutoff distance of any other atoms. The cutoff is 1.0 angstrom for hydrogen/hydrogen collisions, 1.5 angstrom for hydrogen/heavy collisions, and 2.0 angstrom otherwise.

Considering the atoms that are conflicted, PDB2PQR changes selected dihedral angle configurations in increments of 5.0 degrees, looking for positions where the residue does not conflict with other atoms. If modifying a dihedral angle does not result in a debumped configuration then the dihedral angle is reset and the next one is tried. If 10 angles are tried without success the algorithm reports failure.

Warning: It should be noted that this is not an optimal solution. This method is not guaranteed to find a solution if it exists and will accept the first completely debumped state found, not the optimal state.

Additionally, PDB2PQR does not consider water atoms when looking for conflicts.

Hydrogen bond optimization

Unless otherwise indicated with `--noopts`, PDB2PQR will attempt to add hydrogens in a way that optimizes hydrogen bonding.

The hydrogen bonding network optimization seeks, as the name suggests, to optimize the hydrogen bonding network of the biomolecule. Currently this entails manipulating the following residues:

- Flipping the side chains of HIS (including user defined HIS states), ASN, and GLN residues;
- Rotating the sidechain hydrogen on SER, THR, TYR, and CYS (if available);
- Determining the best placement for the sidechain hydrogen on neutral HIS, protonated GLU, and protonated ASP;
- Optimizing all water hydrogens.

Titration state assignment

Versions 2.1 and earlier of PDB2PQR offered the following methods to assign titration states to molecules at a specified pH.

- PROPKA. This method uses the PROPKA software to assign titration states. More information about PROPKA can be found on its website.
- PDB2PKA. Uses a Poisson-Boltzmann method to assign titration states. This approach is loosely related to the method described by Nielsen and Vriend (2001) doi:10.1002/prot.10.

The current version () of PDB2PQR currently only supports PROPKA while we address portability issues in PDB2PKA.

PDB2PQR has the ability to recognize certain protonation states and keep them fixed during optimization. To use this feature manually rename the residue name in the PDB file as follows:

- Neutral ASP: ASH
- Negative CYS: CYM
- Neutral GLU: GLH
- Neutral HIS: HIE or HSE (epsilon-protonated); HID or HSD (delta-protonated)
- Positive HIS: HIP or HSP
- Neutral LYS: LYN
- Negative TYR: TYM

PDB2PQR is unable to assign charges and radii when they are not available in the forcefield - thus this warning message will occur for most ligands unless a `MOL2` file is provided for the ligand with the `--ligand` option. Occasionally this message will occur in error for a standard amino acid residue where an atom or residue may be misnamed. However, some of the protonation states derived from the PROPKA results are not supported in the requested forcefield and thus PDB2PQR is unable to get charges and radii for that state. PDB2PQR currently supports the following states as derived from PROPKA:

Protonation State	AMBER Support	CHARMM Support	PARSE Support
Neutral N-Terminus	No	No	Yes
Neutral C-Terminus	No	No	Yes
Neutral ARG	No	No	No
Neutral ASP	Yes[1]	Yes	Yes
Negative CYS	Yes[1]	No	Yes
Neutral GLU	Yes[1]	Yes	Yes
Neutral HIS	Yes	Yes	Yes
Neutral LYS	Yes[1]	No	Yes
Negative TYR	No	No	Yes

Other software

A variety of other software can be used to visualize and process the results of PDB2PQR and APBS calculations.

Visualization software

Examples of visualization software that work with output from PDB2PQR and APBS:

- PyMOL
- VMD
- Chimera
- PMV

Dynamics simulations

As an example of PDB2PQR and APBS integration with molecular mechanics sofware, the iAPBS library was developed to facilitate the integration of APBS with other molecular simulation packages. This library has enabled the integration of APBS with several molecular dynamics packages, including NAMD, AMBER, and CHARMM.

APBS is also used directly by Brownian dynamics software such as SDA and BrownDye.

2.3 Getting help

2.3.1 GitHub issues

Our preferred mechanism for user questions and feedback is via GitHub issues. We monitor these issues daily and usually respond within a few days.

2.3.2 Announcements

Announcements about updates to the APBS-PDB2PQR software and related news are available through our mailing list; please register for updates.

[1] Only if residue is not a terminal residue; if the residue is terminal it will not be set to this state.

2.3.3 Contacting the authors

If all else fails, feel free to contact nathanandrewbaker@gmail.com.

2.4 Supporting PDB2PQR

2.4.1 Please register as a user!

Please help ensure continued support for APBS-PDB2PQR by registering your use of our software.

2.4.2 Citing our software

If you use PDB2PQR in your research, please cite one or more of the following papers:

- Jurrus E, Engel D, Star K, Monson K, Brandi J, Felberg LE, Brookes DH, Wilson L, Chen J, Liles K, Chun M, Li P, Gohara DW, Dolinsky T, Konecny R, Koes DR, Nielsen JE, Head-Gordon T, Geng W, Krasny R, Wei G-W, Holst MJ, McCammon JA, Baker NA. Improvements to the APBS biomolecular solvation software suite. Protein Sci, 27 (1), 112-128, 2018. https://doi.org/10.1002/pro.3280
- Unni S, Huang Y, Hanson RM, Tobias M, Krishnan S, Li WW, Nielsen JE, Baker NA. Web servers and services for electrostatics calculations with APBS and PDB2PQR. J Comput Chem, 32 (7), 1488-1491, 2011. http://dx.doi.org/10.1002/jcc.21720
- Dolinsky TJ, Czodrowski P, Li H, Nielsen JE, Jensen JH, Klebe G, Baker NA. PDB2PQR: Expanding and upgrading automated preparation of biomolecular structures for molecular simulations. Nucleic Acids Res, 35, W522-5, 2007. http://dx.doi.org/10.1093/nar/gkm276
- Dolinsky TJ, Nielsen JE, McCammon JA, Baker NA. PDB2PQR: an automated pipeline for the setup, execution, and analysis of Poisson-Boltzmann electrostatics calculations. Nucleic Acids Res, 32, W665-7, 2004. http://dx.doi.org/10.1093/nar/gkh381

2.4.3 Supporting organizations

The PDB2PQR authors would like to give special thanks to the supporting organizations behind the APBS and PDB2PQR software:

National Institutes of Health Primary source of funding for APBS via grant GM069702

National Biomedical Computation Resource Deployment and computational resources support from 2002 to 2020

National Partnership for Advanced Computational Infrastructure Funding and computational resources

Washington University in St. Louis Start-up funding

2.5 Extending PDB2PQR

2.5.1 Adding new forcefield parameters

If you are just adding the parameters of a few residues and atoms to an existing forcefield (e.g., AMBER), you can open the forcefield data file distributed with PDB2PQR (`dat/AMBER.DAT`) directly and add your parameters. After the parameter addition, save the force field data file with your changes. You should also update the corresponding

.names file (`dat/AMBER.names`) if your added residue or atom naming scheme is different from the PDB2PQR canonical naming scheme.

2.5.2 Adding an entirely new forcefield

The following steps outline how to add a new force field to PDB2PQR.

You will need to generate a forcefield data file (e.g., `myff.DAT`) and, if your atom naming scheme of the forcefield is different from the PDB2PQR canonical naming scheme, you will also need to provide a names files (`myFF.names`). The format of the names file is described in *PDB2PQR NAMES files*. It is recommended to build your own forcefield data and names files based on existing PDB2PQR `.DAT` and `.names` examples provided with PDB2PQR in the `dat` directory. After finishing your forcefield data file and names file, these can be used with either the command line or the web server versions of PDB2PQR.

2.5.3 Helping with development

Adding new functionality

PDB2PQR welcomes new contributions; the software API is documented in *API Reference*. To contribute code, submit a *pull request* against the master branch in the PDB2PQR repository. Please be sure to run PDB2PQR tests, as described in *Testing*, before submitting new code.

Helping with to-do items

A list of "to-do" items for the code is available in GitHub Issues. A loosely maintained list auto-generated from the documentation is also presented below.

Todo: need to see whether `super().__init__()` should be called

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/aa.py:docstring of pdb2pqr.aa.Amino.__init__, line 3.)

Todo: Determine why this is different than superclass method.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/aa.py:docstring of pdb2pqr.aa.Amino.create_atom, line 3.)

Todo: why is the force field name "WAT" for this?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/aa.py:docstring of pdb2pqr.aa.LIG.__init__, line 3.)

Todo: Why is water in the amino acid module?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/aa.py:docstring of pdb2pqr.aa.WAT, line 3.)

Todo: There is a huge amount of duplicated code in this module.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/aa.py:docstring of pdb2pqr.aa.WAT.create_atom, line 5.)

Todo: This module should be broken into separate files.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/biomolecule.py:d of pdb2pqr.biomolecule, line 3.)

Todo: Since the misslist is used to identify incorrect charge assignments, this routine does not list the 3 and 5 termini of nucleic acid chains as having non-integer charge even though they are (correctly) non-integer.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/biomolecule.py:d of pdb2pqr.biomolecule.Biomolecule.charge, line 3.)

Todo: Figure out if this is redundant with `Biomolecule.num_bio_atoms()`

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/biomolecule.py:d of pdb2pqr.biomolecule.Biomolecule.num_heavy, line 3.)

Todo: This function needs to be cleaned and simplified

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/biomolecule.py:d of pdb2pqr.biomolecule.Biomolecule.set_termini, line 6.)

Todo: Why are we setting residue types to numeric values (see code)?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/biomolecule.py:d of pdb2pqr.biomolecule.Biomolecule.update_residue_types, line 3.)

Todo: Manage several blocks of data.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/cif.py:docstring of pdb2pqr.cif.read_cif, line 3.)

Todo: This class needs to be susbtantially refactored in to multiple classes with clear responsibilities.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/debump.py:docst of pdb2pqr.debump.Debump, line 3.)

Todo: Why are files being loaded so deep in this function?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.__init__, line 3.)

Todo: Should this be a staticmethod or a classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_amber_params, line 3.)

Todo: Figure out why **`residue.type`** has `int` values

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_amber_params, line 5.)

Todo: Should this be a staticmethod or a classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_charmm_params, line 3.)

Todo: Figure out why **`residue.type`** has `int` values

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_charmm_params, line 5.)

Todo: Why do both *`get_params()`* and *`get_params1()`* exist?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_params, line 7.)

Todo: Why do both *`get_params()`* and *`get_params1()`* exist?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_params1, line 7.)

Todo: Should this be a staticmethod or a classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.Forcefield.get_parse_params, line 3.)

Todo: Should this be a staticmethod instead of classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc of pdb2pqr.forcefield.ForcefieldHandler.find_matching_names, line 3.)

Todo: Should this be a staticmethod instead of classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/forcefield.py:doc
of pdb2pqr.forcefield.ForcefieldHandler.update_map, line 4.)

Todo: This module has too many lines and should be simplified.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens, line 5.)

Todo: This class really needs to be refactored.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens.HydrogenRoutines, line 3.)

Todo: Remove hard-coded **`progress`** threshold and increment values.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens.HydrogenRoutines.optimize_hydrogens, line 6.)

Todo: This function needs to be simplified.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens.HydrogenRoutines.optimize_hydrogens, line 10.)

Todo: The type of the **`res`** appears to be incorrect.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens.HydrogenRoutines.parse_hydrogen, line 6.)

Todo: This function is too long and needs to be simplified.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/__init
of pdb2pqr.hydrogens.HydrogenRoutines.parse_hydrogen, line 9.)

Todo: Lots of code in this function could be accelerated with `numpy`.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.get_pair_energy, line 3.)

Todo: Lots of hard-coded parameters in this function that need to be abstracted out.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.get_pair_energy, line 7.)

Todo: Should this be a staticmethod rather than a classmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.get_position_with_three_bonds, line 3.)

Todo: Remove hard-coded values in function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.get_position_with_three_bonds, line 6.)

Todo: Remove some hard-coded values in this function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.get_positions_with_two_bonds, line 3.)

Todo: Remove hard-coded hydrogen bond distance and angles.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.is_hbond, line 3.)

Todo: Does this need to be a classmethod or could it be a staticmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.make_atom_with_one_bond_h, line 3.)

Todo: Does this need to be a classmethod or could it be a staticmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.make_atom_with_one_bond_lp, line 3.)

Todo: Does this need to be a classmethod or could it be a staticmethod?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.make_water_with_one_bond, line 5.)

Todo: Remove some hard-coded values in this function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.try_single_alcoholic_h, line 3.)

Todo: Remove some hard-coded values in this function and figure out where others (e.g., “72”) come from.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/optim
of pdb2pqr.hydrogens.optimize.Optimize.try_single_alcoholic_lp, line 3.)

Todo: Replace hard-coded values in the function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/structu
of pdb2pqr.hydrogens.structures.Alcoholic.finalize, line 3.)

Todo: Rename this and related methods to conform with PEP8

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/structu
of pdb2pqr.hydrogens.structures.HydrogenHandler.endElement, line 3.)

Todo: Rename this and related methods to conform with PEP8

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/structu
of pdb2pqr.hydrogens.structures.HydrogenHandler.startElement, line 3.)

Todo: This looks like a bug: donorh ends up being the last item in donor.bonds. This may be fixed by setting a best_donorh to go with bestdist and using best_donorh in the function below

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/hydrogens/structu
of pdb2pqr.hydrogens.structures.Water.try_acceptor, line 3.)

Todo: Remove hard-coded parameters.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/inputgen.py:docs
of pdb2pqr.inputgen.Elec.__init__, line 3.)

Todo: is this function still useful?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/inputgen.py:docs
of pdb2pqr.inputgen.Input.dump_pickle, line 3.)

Todo: This should be a context manager (to close the open file).

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/io.py:docstring
of pdb2pqr.io.get_pdb_file, line 7.)

Todo: Remove hard-coded parameters.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/io.py:docstring
of pdb2pqr.io.get_pdb_file, line 8.)

Todo: This function should be moved into the APBS code base.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/io.py:docstring
of pdb2pqr.io.read_dx, line 9.)

Todo: This function should be moved into the APBS code base.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/io.py:docstring of pdb2pqr.io.write_cube, line 6.)

Todo: Some of the definitions in this module belong in a configuration file other than here.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/ligand/__init__.p of pdb2pqr.ligand, line 3.)

Todo: It seems inconsistent that this function pulls radii from a dictionary and the biomolecule routines use force field files.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/ligand/mol2.py:d of pdb2pqr.ligand.mol2.Mol2Atom.assign_radius, line 3.)

Todo: Need separate argparse groups for PDB2PKA and PROPKA. These exist but need real options.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.build_main_parser, line 3.)

Todo: this module is already too long but this function fits better here. Other possible place would be utilities.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.drop_water, line 3.)

Todo: This function should be moved into the APBS code base.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.dx_to_cube, line 8.)

Todo: These routines should be generalized to biomolecules; none of them are specific to biomolecules.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.non_trivial, line 3.)

Todo: Move this to another module (io)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.print_pdb, line 3.)

Todo: Move this to another module (io)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.print_pqr, line 3.)

Todo: I wish this could be done with argparse.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/main.py:docstrin of pdb2pqr.main.transform_arguments, line 3.)

Todo: This code is duplicated in several places.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/na.py:docstring of pdb2pqr.na.Nucleic.create_atom, line 5.)

Todo: If multiple modifications are present, only the last one in the file is preserved.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/pdb.py:docstring of pdb2pqr.pdb.REVDAT.__init__, line 3.)

Todo: This code could be combined with `inputgen`.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin of pdb2pqr.psize, line 3.)

Todo: This code should be moved to the APBS code base.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin of pdb2pqr.psize, line 5.)

Todo: This is messed up. Why are we parsing the PQR manually here when we already have other routines to do that? This function should be replaced by a call to existing routines.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin of pdb2pqr.psize.Psize.parse_lines, line 3.)

Todo: remove hard-coded values from this function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin of pdb2pqr.psize.Psize.set_fine_grid_points, line 3.)

Todo: Replace hard-coded values in this function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin of pdb2pqr.psize.Psize.set_length, line 3.)

Todo: Remove hard-coded values from this function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/psize.py:docstrin
of pdb2pqr.psize.Psize.set_smallest, line 9.)

Todo: There are many unnecessary parameters in this module due to FORTRAN/C assumptions about how the code should behave.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/quatfit.py:docstri
of pdb2pqr.quatfit, line 13.)

Todo: Remove hard-coded parameters of function.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/quatfit.py:docstri
of pdb2pqr.quatfit.qfit, line 4.)

Todo: Should this class have a member variable for dihedrals? Pylint complains!

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pdb2pqr/checkouts/v3.1.0/pdb2pqr/residue.py:docstr
of pdb2pqr.residue.Residue, line 3.)

2.6 File formats

2.6.1 Molecular structure formats

PQR molecular structure format

This format is a modification of the PDB format which allows users to add charge and radius parameters to existing PDB data while keeping it in a format amenable to visualization with standard molecular graphics programs. The origins of the PQR format are somewhat uncertain, but has been used by several computational biology software programs, including MEAD and AutoDock. UHBD uses a very similar format called QCD.

APBS reads very loosely-formatted PQR files: all fields are whitespace-delimited rather than the strict column formatting mandated by the PDB format. This more liberal formatting allows coordinates which are larger/smaller than ± 999 Å. APBS reads data on a per-line basis from PQR files using the following format::

```
Field_name Atom_number Atom_name Residue_name Chain_ID Residue_number X Y Z Charge␣
↪Radius
```

where the whitespace is the most important feature of this format. The fields are:

`Field_name` A string which specifies the type of PQR entry and should either be ATOM or HETATM in order to be parsed by APBS.

`Atom_number` An integer which provides the atom index.

`Atom_name` A string which provides the atom name.

`Residue_name` A string which provides the residue name.

Chain_ID An optional string which provides the chain ID of the atom. Note that chain ID support is a new feature of APBS 0.5.0 and later versions.

Residue_number An integer which provides the residue index.

X Y Z 3 floats which provide the atomic coordinates (in Å)

Charge A float which provides the atomic charge (in electrons).

Radius A float which provides the atomic radius (in Å).

Clearly, this format can deviate wildly from PDB due to the use of whitespaces rather than specific column widths and alignments. This deviation can be particularly significant when large coordinate values are used. However, in order to maintain compatibility with most molecular graphics programs, the PDB2PQR program and the utilities provided with APBS attempt to preserve the PDB format as much as possible.

PDB molecular structure format

The PDB file format is described in detail in the Protein Data Bank documentation.

MOL2 molecular structure format

The MOL2 file format is a popular method for specifying chemical structure, including atom types, positions, and bonding. It is described in detail in the Tripos documentation.

2.6.2 Parameter formats

PDB2PQR NAMES files

Much of the difficulty in adding a new forcefield to PDB2PQR depends on the naming scheme used in that forcefield.

XML file format

To start, either a flat file or XML file containing the desired forcefield's parameters should be made - see `AMBER.DAT` and `AMBER.xml` for examples. If the forcefield's naming scheme matches the canonical naming scheme, that's all that is necessary. If the naming schemes differ, however, conversions must be made. These are made in the `*.names` file (see `CHARMM.names`, for example). In this file you will see sections like:</p>

```
<residue>
  <name>WAT</name>
  <useresname>TP3M</useresname>
  <atom>
    <name>O</name>
    <useatomname>OH2</useatomname>
  </atom>
</residue>
```

This section tells PDB2PQR that for the oxygen atom O in WAT, CHARMM uses the names OH2 and TP3M, respectively. When the XML file is read in, PDB2PQR ensures that the WAT/O pair points to TP3M/OH2 such that the appropriate parameters are returned. But for naming schemes that greatly differ from the PDB2PQR canonical naming scheme, this could get really ugly. As a result, PDB2PQR can use regular expressions to simplify the renaming process, i.e.:

```
<residue>
  <name>[NC]?...$</name>
  <atom>
    <name>H</name>
    <useatomname>HN</useatomname>
  </atom>
</residue>
```

This section of code will ensure that the H atom of all canonical residue names that match the `[NC]?...$` regular expression point to HN instead. This regular expression matches all three-letter residue names, residue names with an 'N' prepended (N-Termini), and residue names with a 'C' prepended (C-Termini). For twenty amino acids, sixty residue name changes can all be done by a single section. The use of regular expressions is therefore a much more powerful method of handling naming scheme differences than working on a one to one basis.

There are a few other additional notes when using the `.names` file. First, the `$group` variable is used to denote the matching group of a regular expression, for instance:

```
<residue>
  <name>HI([PDE])$</name>
  <useresname>HS$group</useresname>
</residue>
```

This section replaces HIP/HID/HIE with HSP/HSD/HSE by first matching the HI([PDE])$ regular expression and then using the group that is enclosed by parantheses to fill in the name to use.

Second, sections are cumulative - since CHARMM, for instance, has a patch-based naming scheme, one single canonical residue name can map to multiple forcefield-scheme names. Let's look at how to map an SS-bonded Cysteine (canonical name CYX) to the CHARMM naming scheme:

```
<residue>
  <name>CYX</name>
  <useresname>CYS</useresname>
</residue>
<residue>
  <name>CYX</name>
  <useresname>DISU</useresname>
  <atom>
    <name>CB</name>
    <useatomname>1CB</useatomname>
  </atom>
  <atom>
    <name>SG</name>
    <useatomname>1SG</useatomname>
  </atom>
</residue>
```

The CYX residue is first mapped to CHARMM's CYS, and then to CHARMM's DISU object. All atom names that are found in DISU overwrite those found in CYS - in effect, the DISU patch is applied to CYS, yielding the desired CYX. This cumulative can be repeated as necessary.

Caveats about atom naming

In an ideal world each individual residue and atom would have a standard, distinct name. Unfortunately several naming schemes for atoms exist, particularly for hydrogens. As such, in order to detect the presence/absence of atoms in a biomolecule, an internal canonical naming scheme is used. The naming scheme used in PDB2PQR is the one recommended by the PDB itself, and derives from the IUPAC naming recommendations[1]

This canonical naming scheme is used as the default PDB2PQR output. All conversions in PDB2PQR use the internal canonical naming scheme to determine distinct atom names. In previous versions of PDB2PQR, these conversions were stored in long lists of if statements, but for transparency and editing this is a bad thing. Instead, all conversions can now be found in XML as described above.

There are a few additions to the canonical naming scheme, mirrored after the AMBER naming scheme (chosen since for the most part it follows the IUPAC recommendations). These changes are made in `PATCHES.xml`, and allow any of the following to be patched as necessary as well as detected on input:

N* N-Terminal Residue (i.e. NALA, NLEU)

NEUTRAL-N* Neutral N-Terminal Residue

C* C-Terminal Residue (i.e. CLYS, CTYR)

NEUTRAL-C* Neutral C-Terminal Residue

***5** 5-Terminus for Nucleic Acids (i.e. DA5)

***3** 3-Terminus for Nucleic Acids (i.e. DA3)

ASH Neutral ASP

CYX SS-bonded CYS

CYM Negative CYS

GLH Neutral GLU

HIP Positive HIS

HID Neutral HIS, proton HD1 present

HIE Neutral HIS, proton HE2 present

LYN Neutral LYS

TYM Negative TYR

PDB2PQR DAT files

A PDB2PQR `.DAT` file has the following whitespace-delimited columns:

1. Name of the residue (`str`, e.g., RU, ARG, WAT, etc.)
2. Name of the atom (`str`, e.g., HB3, CA, HA, etc.)
3. Charge of the atom (`float`, in electrons)
4. Radius of the atom (`float`, in Å)
5. (optional) Atom type (`str`, e.g., HT, etc.)

Lines that start with # are treated as comments.

[1] J. L. Markley, et al., "Recommendations for the Presentation of NMR Structures of Proteins and Nucleic Acids," Pure & Appl. Chem., 70 (1998): 117-142. DOI:10.1046/j.1432-1327.1998.2560001.x

2.7 API Reference

The **`pdb2pqr30`** command provides a command-line interface to PDB2PQR's functionality. It is built on classes and functions in the *`pdb2pqr`* module. The API of *`pdb2pqr`* is documented here for developers who might want to directly use the PDB2PQR code.

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

2.7.1 Forcefield support modules

`definitions`

XML handling for biomolecular residue topology definitions.

Code author: Jens Erik Nielsen

Code author: Todd Dolinsky

Code author: Yong Huang

class `pdb2pqr.definitions.`**`Definition`**(*aa_file*, *na_file*, *patch_file*)

Force field topology definitions.

The Definition class contains the structured definitions found in the files and several mappings for easy access to the information.

`__init__`(*aa_file*, *na_file*, *patch_file*)

Initialize object.

Parameters

- **`aa_file`** (*`file`*) – file-like object with amino acid definitions
- **`na_file`** (*`file`*) – file-like object with nucleic acid definitions
- **`patch_file`** (*`file`*) – file-like object with patch definitions

`add_patch`(*patch*, *refname*, *newname*)

Add a patch to a topology definition residue.

Parameters

- **`patch`** (`Patch`) – the patch object to add
- **`refname`** (*`str`*) – the name of the object to add the patch to
- **`newname`** (*`str`*) – the name of the new (patched) object

class `pdb2pqr.definitions.`**`DefinitionAtom`**(*name=None*, *x=None*, *y=None*, *z=None*)

Store force field atom topology definitions.

`__init__`(*name=None*, *x=None*, *y=None*, *z=None*)

Initialize class.

Parameters

- **`name`** (*`str`*) – atom name
- **`x`** (*`float`*) – x-coordinate
- **`y`** (*`float`*) – y-coordinate

- **z** (*float*) – z-coordinate

is_backbone
Identify whether atom is in backbone.

Returns true if atom name is in backbone, otherwise false

Return type bool

class pdb2pqr.definitions.**DefinitionHandler**
Handle definition XML file content.

__init__()
Initialize self. See help(type(self)) for accurate signature.

characters(*text*)
Parse text data in XML.

Parameters **text** (*str*) – text data to parse

endElement(*name*)
End XML element parsing.

Parameters **name** (*str*) – element name

Raises

- **KeyError** – for invalid atom or residue names
- **RuntimeError** – for unexpected XML features or format

startElement(*name*, *_*)
Start XML element parsing.

Parameters **name** (*str*) – element name

class pdb2pqr.definitions.**DefinitionResidue**
Force field toplogy representation for a residue.

__init__()
Initialize the class

Parameters **atoms** (*list*) – list of atom-like (HETATM or ATOM) objects to be stored

get_nearest_bonds(*atomname*)
Get bonded atoms near a given atom.

Parameters **atomname** (*str*) – name of specific atom

Returns list of nearby bonded atom names

Return type [str]

class pdb2pqr.definitions.**Patch**
Residue patches for structure topologies.

__init__()
Initialize self. See help(type(self)) for accurate signature.

forcefield

Force fields are fun.

The forcefield structure is modeled off of the structures.py file, where each forcefield is considered a chain of residues of atoms.

Code author: Todd Dolinsky

Code author: Yong Huang

class pdb2pqr.forcefield.**Forcefield**(*ff_name*, *definition*, *userff*, *usernames=None*)

Parameter definitions for a given forcefield.

Note: Pass ff and ff names file like objects rather than sorting out whether to use user-created files here.

The forcefield class contains definitions for a given forcefield. Each forcefield object contains a dictionary of residues, with each residue containing a dictionary of atoms. Dictionaries are used instead of lists as the ordering is not important. The forcefield definition files are unedited, directly from the forcefield - all transformations are done within.

__init__(*ff_name*, *definition*, *userff*, *usernames=None*)

Initialize the class by parsing the definition file.

Todo: Why are files being loaded so deep in this function?

Parameters

- **ff_name** (*str*) – the name of the forcefield (can be None)
- **definition** (Definition) – the definition object for the Forcefield
- **userff** (*str*) – path for user-defined forcefields file
- **usernames** (*str*) – path to user-defined atom/residue names file

Raises **ValueError** – if invalid force field names specified

classmethod get_amber_params(*residue*, *name*)

Get AMBER forcefield definitions.

Todo: Should this be a staticmethod or a classmethod?

Todo: Figure out why **residue.type** has int values

Parameters

- **residue** (Residue) – the residue
- **name** (*str*) – the atom name

Returns (forcefield name of residue, forcefield name of atom)

Return type (str, str)

classmethod get_charmm_params(*residue*, *name*)

Get CHARMM forcefield definitions.

Todo: Should this be a staticmethod or a classmethod?

Todo: Figure out why **`residue.type`** has `int` values

Parameters

- **`residue`** (`Residue`) – the residue
- **`name`** (*`str`*) – the atom name

Returns (forcefield name of residue, forcefield name of atom)

Return type (str, str)

`get_group`(*resname*, *atomname*)

Get the group/type associated with the input fields.

Parameters

- **`resname`** (*`str`*) – the residue name
- **`atomname`** (*`str`*) – the atom name

Returns group name or empty string

Return type str

`get_names`(*resname*, *atomname*)

Get forcefield names associated with residue and atom.

The names passed in point to ForcefieldResidue and ForcefieldAtom objects which may have different names; grab these names and return.

Parameters

- **`resname`** (*`str`*) – the residue name
- **`atomname`** (*`str`*) – the atom name

Returns (forcefield's name for this residue, forcefield's name for this atom)

Return type (str, str)

`get_params`(*resname*, *atomname*)

Get the charge and radius parameters for an atom in a residue.

The residue itself is needed instead of simply its name because the forcefield may use a different residue name than the standard amino acid name.

Todo: Why do both *`get_params()`* and *`get_params1()`* exist?

Parameters

- **`resname`** (*`str`*) – the residue name
- **`atomname`** (*`str`*) – the atom name

Returns (charge of the atom, radius of the atom)

Return type (float, float)

get_params1(*residue*, *name*)

Get the charge and radius parameters for an atom in a residue.

The residue itself is needed instead of simply its name because the forcefield may use a different residue name than the standard amino acid name.

Todo: Why do both `get_params()` and `get_params1()` exist?

Parameters

- **resname** (`str`) – the residue name
- **name** (`str`) – the atom name

Returns (charge of the atom, radius of the atom)

Return type (float, float)

classmethod get_parse_params(*residue*, *name*)

Get PARSE forcefield definitions.

Todo: Should this be a staticmethod or a classmethod?

Parameters

- **residue** (`Residue`) – the residue
- **name** (`str`) – the atom name

Returns (forcefield name of residue, forcefield name of atom)

Return type (str, str)

get_residue(*resname*)

Return the residue object with the given resname.

Parameters **resname** (`str`) – the name of the residue

Returns residue object

Return type *ForcefieldResidue*

has_residue(*resname*)

Check if the residue name is in the map or not.

Parameters **resname** (`str`) – the residue name to search

Returns indication of whether resname is in map

Return type bool

class pdb2pqr.forcefield.**ForcefieldAtom**(*name*, *charge*, *radius*, *resname*, *group=''*)

ForcefieldAtom class.

Contains fields that are related to the forcefield at the atom level.

__init__(*name*, *charge*, *radius*, *resname*, *group=''*)

Initialize the object.

Parameters

- **name** (*str*) – atom name
- **charge** (*float*) – the charge on the atom
- **radius** (*float*) – the radius of the atom
- **resname** (*str*) – the residue name
- **group** (*str*) – the group name

get(*name*)
Get a member of the ForcefieldAtom class.

Parameters **name** (*str*) – the name of the member, including:

- name: The atom name (returns string)
- charge: The charge on the atom (returns float)
- radius: The radius of the atom (returns float)
- epsilon: The epsilon assocaited with the atom (returns float)

Returns the value of the member

Raises **KeyError** – if member does not exist

class pdb2pqr.forcefield.**ForcefieldHandler**(*map_*, *reference*)
Process XML-format force field parameter files.

__init__(*map_*, *reference*)
Initialize handler

Parameters

- **map** (*dict*) – dictionary of paramaeter information
- **reference** (*dict*) – reference map for force field

characters(*text*)
Parse text information within XML tag.

Parameters **text** – the text value between the XML tags

endElement(*name*)
End XML element parsing.

Parameters **name** (*str*) – the name of the element

classmethod find_matching_names(*regname*, *map_*)
Find strings in the map that match the given regular expression.

Todo: Should this be a staticmethod instead of classmethod?

Parameters

- **regname** (*str*) – the regular expression to search for in the map
- **map** (*dict*) – the dictionary to search

Returns a list of regular expression match objects for dictionary keys that match the regular expression.

Return type [re.Match]

startElement(*name*, _)
Start XML element parsing.

Parameters **name** (*str*) – element name

classmethod update_map(*toname*, *fromname*, *map_*)
Update the given map by adding a pointer from a new name to an object.

Todo: Should this be a staticmethod instead of classmethod?

Parameters

- **toname** (*str*) – the new name for the object
- **fromname** (*str*) – the old name for the object
- **map** (*dict*) – a dictionary of forcefield-related items

class pdb2pqr.forcefield.**ForcefieldResidue**(*name*)
ForcefieldResidue class

The ForceFieldResidue class contains a mapping of all atoms within the residue for easy searching.

__init__(*name*)
Initialize the ForceFieldResidue object.

Parameters **name** (*str*) – the name of the residue

add_atom(*atom*)
Add an atom to the ForcefieldResidue object.

Parameters **atom** (Atom) – the atom to be added

get_atom(*atomname*)
Return the atom object with the given atomname.

Parameters **resname** (*str*) – the name of the atom

Returns the atom object

Return type *ForcefieldAtom*

has_atom(*atomname*)
Check to see if the named atom is in the current residue.

Parameters **atomname** (*str*) – the name of the atom to search for

Returns indication of whether atmo is present

Return type bool

2.7.2 Molecular structure modules

aa

Amino Acid Structures for PDB2PQR

This module contains the base amino acid structures for pdb2pqr.

Code author: Todd Dolinsky

Code author: Nathan Baker

class pdb2pqr.aa.**ALA**(*atoms*, *ref*)
Alanine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**ARG**(*atoms*, *ref*)
Arginine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

set_state()
Set forcefield name based on current titration state.

class pdb2pqr.aa.**ASN**(*atoms*, *ref*)
Asparagine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**ASP**(*atoms*, *ref*)
Aspartic acid class.

`__init__`(*atoms*, *ref*)
Initialize object.

Parameters

- **`atoms`** (*`[Atom]`*) – A list of `Atom` objects to be stored in this object
- **`ref`** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

`set_state`()
Set forcefield name based on current titration state.

class `pdb2pqr.aa.`**`Amino`**(*atoms*, *ref*)
Amino acid class

This class provides standard features of the amino acids.

`__init__`(*atoms*, *ref*)
Initialize object.

Todo: need to see whether `super().__init__()` should be called

Parameters

- **`atoms`** (*`[Atom]`*) – A list of `Atom` objects to be stored in this object
- **`ref`** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`add_atom`(*atom*)
Add atom to residue.

Override the existing add_atom; include the link to the reference object.

Parameters **`atom`** (`Atom`) – atom to add

`add_dihedral_angle`(*value*)
Add a dihedral angle to the residue list.

Parameters **`value`** (*`float`*) – dihedral angle (in degrees) to add

`create_atom`(*atomname*, *newcoords*)
Create an atom.

Todo: Determine why this is different than superclass method.

Override the generic residue's version of create_atom().

Parameters

- **`atomname`** (*`str`*) – name of atom
- **`newcoords`** (*`[float, float, float]`*) – new coordinates for atom

rebuild_tetrahedral(*atomname*)
Rebuild a tetrahedral hydrogen group.

This is necessary due to the shortcomings of the quatfit routine - given a tetrahedral geometry and two existing hydrogens, the quatfit routines have two potential solutions. This function uses basic tetrahedral geometry to fix this issue.

Parameters **atomname** (*str*) – the atom name to add

Returns indication of whether this was successful

Return type bool

set_state()
Set the name to use for the forcefield based on the current state.

Uses `N*` and `C*` for termini.

class pdb2pqr.aa.**CYS**(*atoms*, *ref*)
Cysteine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

set_state()
Set forcefield name based on current state.

If SS-bonded, use CYX. If negatively charged, use CYM. If HG is not present, use CYX.

class pdb2pqr.aa.**GLN**(*atoms*, *ref*)
Glutamine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**GLU**(*atoms*, *ref*)
Glutamic acid class.

`__init__`(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (`[Atom]`) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

`set_state`()
Set forcefield name based on current titration state.

class `pdb2pqr.aa.`**`GLY`**(*atoms*, *ref*)
Glycine class.

`__init__`(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (`[Atom]`) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class `pdb2pqr.aa.`**`HIS`**(*atoms*, *ref*)
Histidine class.

`__init__`(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (`[Atom]`) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

`set_state`()
Set forcefield name based on current titration state.

Histidines are a special case due to the presence of several different forms. This function sets all neutral forms of HIS to neutral HIS by checking to see if optimization removed **`hacceptor`** or **`hdonor`** flags. Otherwise HID is used as the default.

class pdb2pqr.aa.**ILE**(*atoms*, *ref*)
Isoleucine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**LEU**(*atoms*, *ref*)
Leucine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**LIG**(*atoms*, *ref*)
Generic ligand class.

__init__(*atoms*, *ref*)
Initialize this object.

Todo: why is the force field name “WAT” for this?

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the residue. Used to convert from the alternate naming scheme to the main naming scheme.

add_atom(*atom*)
Add an atom to the residue.

Override the existing add_atom - include the link to the reference object.

Parameters **atom** (Atom) – add atom to residue

create_atom(*atomname*, *newcoords*)
Create a ligand atom.

Parameters

- **atomname** (*str*) – name of atom to be added
- **newcoords** (*[float, float, float]*) – coordinates for new atom

class pdb2pqr.aa.**LYS**(*atoms*, *ref*)
Lysine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

set_state()
Set forcefield name based on current titration state.

class pdb2pqr.aa.**MET**(*atoms*, *ref*)
Methionine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**PHE**(*atoms*, *ref*)
Phenylalanine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class `pdb2pqr.aa.`**PRO**(*atoms*, *ref*)

Proline class.

`__init__`(*atoms*, *ref*)

Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()

Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

`set_state`()

Set forcefield name based on the current state.

Uses `N*` and `C*` for termini.

class `pdb2pqr.aa.`**SER**(*atoms*, *ref*)

Serine class.

`__init__`(*atoms*, *ref*)

Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()

Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class `pdb2pqr.aa.`**THR**(*atoms*, *ref*)

Threonine class.

`__init__`(*atoms*, *ref*)

Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

`letter_code`()

Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**TRP**(*atoms*, *ref*)
Tryptophan class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**TYR**(*atoms*, *ref*)
Tyrosine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

set_state()
Set forcefield name based on current titration state.

class pdb2pqr.aa.**VAL**(*atoms*, *ref*)
Valine class.

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of Atom objects to be stored in this object
- **ref** (Residue) – The reference object for the amino acid. Used to convert from the alternate naming scheme to the main naming scheme.

letter_code()
Return letter code for amino acid.

Returns amino acid 1-letter code

Return type str

class pdb2pqr.aa.**WAT**(*atoms*, *ref*)
Water class.

Todo: Why is water in the amino acid module?

__init__(*atoms*, *ref*)
Initialize object.

Parameters

- **atoms** (*[Atom]*) – A list of `Atom` objects to be stored in this object
- **ref** (`Residue`) – The reference object for the residue. Used to convert from the alternate naming scheme to the main naming scheme.

add_atom(*atom*)
Add an atom to the residue.

Override the existing add_atom - include the link to the reference object.

Parameters **atom** (`Atom`) – add atom to residue

create_atom(*atomname*, *newcoords*)
Create a water atom.

Note the HETATM field.

Todo: There is a huge amount of duplicated code in this module.

Parameters

- **atomname** (*str*) – name of atom to be added
- **newcoords** (*[float, float, float]*) – coordinates for new atom

water_residue_names = ['HOH', 'WAT']

hydrogens and submodule contents

hydrogens

Hydrogen optimization module for PDB2PQR.

This is an module for hydrogen optimization routines.

Todo: This module has too many lines and should be simplified.

Code author: Todd Dolinsky

Code author: Jens Erik Nielsen

Code author: Yong Huang

Code author: Nathan Baker

class pdb2pqr.hydrogens.**HydrogenRoutines**(*debumper*, *handler*)

The main routines for hydrogen optimization.

Todo: This class really needs to be refactored.

__init__(*debumper*, *handler*)

Initialize object.

Parameters

- **debumper** (debump.Debump) – Debump object
- **handler** (HydrogenHandler) – HydrogenHandler object

cleanup()

Delete extra carboxylic atoms.

If there are any extra carboxlyic `*1` atoms, delete them. This may occur when no optimization is chosen.

initialize_full_optimization()

Initialize the full optimization.

Detects all optimizeable donors and acceptors and sets the internal optlist.

initialize_wat_optimization()

Initialize optimization for waters only.

Detects all optimizeable donors and acceptors and sets the internal optlist.

is_optimizeable(*residue*)

Check to see if the given residue is optimizeable. There are three ways to identify a residue:

1. By name (i.e., HIS)
2. By reference name - a PDB file HSP has a HIS reference name
3. By patch - applied by **propka**, terminal selection

Parameters **residue** (Residue) – the residue in question

Returns None if not optimizeable, otherwise the OptimizationHolder instance that corresponds to the residue.

Return type None or *OptimizationHolder*

optimize_hydrogens()

The main driver for the optimization.

Note: Should be called only after the optlist has been initialized.

Todo: Remove hard-coded **progress** threshold and increment values.

Todo: This function needs to be simplified.

parse_hydrogen(*res*, *topo*)

Parse a list of lines in order to make a hydrogen definition.

This is the current definition: `Name Ttyp A R # Stdconf HT Chi OPTm`

Todo: The type of the **`res`** appears to be incorrect.

Todo: This function is too long and needs to be simplified.

Parameters

- **`res`** (*`unknown`*) – the lines to parse (list)
- **`topo`** (`pdb2pqr.topology.Topology`) – Topology object

Returns the hydrogen definition object

Return type *HydrogenDefinition*

`classmethod pka_switchstate`(*amb*, *state_id_*)

Switch a residue to a new state by first removing all hydrogens. This routine is used in pKa calculations only!

Parameters

- **`amb`** (*`tup`*) – the amibiguity to switch
- **`state_id`** (*`int`*) – the state id to switch to

`read_hydrogen_def`(*topo*)

Read the hydrogen definition file

Parameters **`topo`** (*`Topology object`*) – Topology object

`set_optimizeable_hydrogens`()

Set any hydrogen listed in HYDROGENS.xml that is optimizeable.

Used BEFORE hydrogen optimization to label atoms so that they won't be debumped - i.e. if SER HG is too close to another atom, don't debump but wait for optimization.

Note: This function should not be used if full optimization is not taking place.

`switchstate`(*states*, *amb*, *state_id*)

Switch a residue to a new state by first removing all hydrogens.

Parameters

- **`states`** (*`list`*) – the list of states
- **`amb`** (*`tup`*) – the amibiguity to switch
- **`state_id`** (*`int`*) – the state id to switch to

`pdb2pqr.hydrogens.`**`create_handler`**(*hyd_path='HYDROGENS.xml'*)

Create and populate a hydrogen handler.

Parameters **`hyd_def_file`** (*`string or pathlib.Path object`*) – path to hydrogen definition file

Returns HydrogenHandler object

Return type *HydrogenHandler*

`hydrogens.optimize`

Hydrogen optimization routines.

Code author: Todd Dolinsky

Code author: Jens Erik Nielsen

Code author: Yong Huang

Code author: Nathan Baker

class `pdb2pqr.hydrogens.optimize.`**`OptimizationHolder`**

A holder class for the XML parser.

`__init__`()

Initialize self. See help(type(self)) for accurate signature.

class `pdb2pqr.hydrogens.optimize.`**`Optimize`**

The holder class for the hydrogen optimization routines.

Individual optimization types inherit off of this class. Any functions used by multiple types appear here.

`__init__`()

Initialize self. See help(type(self)) for accurate signature.

static get_hbond_angle(*atom1*, *atom2*, *atom3*)

Get the angle between three atoms

Parameters

- **atom1** (`Atom`) – the first atom
- **atom2** (`Atom`) – the second (vertex) atom
- **atom3** (`Atom`) – the third atom

Returns the angle between the atoms in degrees

Return type float

static get_pair_energy(*donor*, *acceptor*)

Get the energy between two atoms

Todo: Lots of code in this function could be accelerated with `numpy`.

Todo: Lots of hard-coded parameters in this function that need to be abstracted out.

Parameters

- **donor** (`Atom`) – the first atom in the pair
- **acceptor** (`Atom`) – the second atom in the pair

Returns the energy of the pair

Return type float

classmethod get_position_with_three_bonds(*atom*)

Find position for last bond in tetrahedral geometry.

Todo: Should this be a staticmethod rather than a classmethod?

Todo: Remove hard-coded values in function.

If there's three bonds in a tetrahedral geometry, there's only one available position. Find that position.

Parameters **atom** (Atom) – atom to check

Returns coordinates

Return type (float, float, float)

classmethod get_positions_with_two_bonds(*atom*)

Return possible coordinates for new bonds.

Todo: Remove some hard-coded values in this function.

Given a tetrahedral geometry with two existing bonds, return the two potential sets of coordinates that are possible for a new bond.

Parameters **atom** (Atom) – atom to test for bonds

Returns 2-tuple of coordinates (floats)

Return type tuple

is_hbond(*donor*, *acc*)

Determine whether this donor acceptor pair is a hydrogen bond.

Todo: Remove hard-coded hydrogen bond distance and angles.

Parameters

- **donor** – donor atom
- **acc** – acceptor atom

Returns whether this pair is a hydrogen bond

Return type bool

make_atom_with_no_bonds(*atom*, *closeatom*, *addname*)

Create an atom with no bonds.

Called for water oxygen atoms with no current bonds. Uses the closeatom to place the new atom directly colinear with the atom and the closeatom.

Parameters

- **atom** (Atom) – the oxygen atom of the water
- **closeatom** (Atom) – the nearby atom (donor/acceptor)
- **addname** (*str*) – the name of the atom to add

classmethod make_atom_with_one_bond_h(*atom*, *addname*)

Add a hydrogen to an alcoholic donor with one existing bond.

Todo: Does this need to be a classmethod or could it be a staticmethod?

Parameters

- **atom** (Atom) – existing atom
- **addname** (*str*) – name of atom to add

classmethod make_atom_with_one_bond_lp(*atom*, *addname*)

Add a lone pair to an alcoholic donor with one existing bond.

Todo: Does this need to be a classmethod or could it be a staticmethod?

Parameters

- **atom** (Atom) – existing atom
- **addname** (*str*) – name of atom to add

classmethod make_water_with_one_bond(*atom*, *addname*)

Add an atom to a water residue that already has one bond.

Uses the water reference structure to align the new atom.

Todo: Does this need to be a classmethod or could it be a staticmethod?

Parameters

- **atom** (Atom) – existing atom
- **addname** (*str*) – name of atom to add

try_positions_three_bonds_h(*donor*, *acc*, *newname*, *loc*)

Try making a hydrogen bond with the lone available position.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor
- **newname** (*str*) – name for new atom

Returns indication of whether atom was added

Return type bool

try_positions_three_bonds_lp(*acc*, *donor*, *newname*, *loc*)

Make a hydrogen bond in the only position possible.

Try making a hydrogen bond using the lone available hydrogen position.

Parameters

- **acc** (Atom) – hydrogen bond acceptor

- **donor** (Atom) – hydrogen bond donor
- **newname** (*str*) – new atom name
- **loc** (*(float, float, float)*) – location for atom (coordinates)

Returns indication whether atom was added

Return type bool

try_positions_with_two_bonds_h(*donor*, *acc*, *newname*, *loc1*, *loc2*)

Try adding a new hydrogen to the two potential locations. If both form hydrogen bonds, place at whatever returns the best bond as determined by get_pair_energy.

Parameters

- **donor** (Atom) – donor atom
- **acc** (Atom) – acceptor atom
- **newname** (*str*) – new atom name
- **loc1** (*(float, float, float)*) – first coordinate to try
- **loc2** (*(float, float, float)*) – first coordinate to try

Returns indication whether atom was added

Return type bool

try_positions_with_two_bonds_lp(*acc*, *donor*, *newname*, *loc1*, *loc2*)

Attempt to place a LP (lone pair) on an atom.

Try placing an LP on a tetrahedral geometry with two existing bonds. If this isn't a hydrogen bond it can return - otherwise ensure that the H(D)-A-LP angle is minimized.

Parameters

- **acc** (Atom) – hydrogen bond acceptor
- **donor** (Atom) – hydrogen bond donor
- **newname** (*str*) – name for lone pair
- **loc1** (*(float, float, float)*) – first location to try
- **loc2** (*(float, float, float)*) – second location to try

Returns indication of whether LP was added

Return type bool

try_single_alcoholic_h(*donor*, *acc*, *newatom*)

Attempt to add an atom to make a hydrogen bond.

Todo: Remove some hard-coded values in this function.

After a new bond has been added using `makeAtomWithOneBond()`, try to find the best orientation by rotating to form a hydrogen bond. If a bond cannot be formed, remove the **newatom** (thereby returning to a single bond).

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

- **newatom** (Atom) – new atom to add

Returns indication whether atom was added

Return type bool

try_single_alcoholic_lp(*acc*, *donor*, *newatom*)
Attempt to add an atom to make a hydrogen bond.

Todo: Remove some hard-coded values in this function and figure out where others (e.g., "72") come from.

After a new bond has been added using `makeAtomWithOneBond()`, ensure that a hydrogen bond has been made. If so, try to minimze the H(D)-A-LP angle. If that cannot be minimized, ignore the bond and remove the atom.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor
- **newatom** (Atom) – new atom to add

Returns indication whether atom was added

Return type bool

hydrogens.structures

Topology-related classes for hydrogen optimization.

class pdb2pqr.hydrogens.structures.**Alcoholic**(*residue*, *optinstance*, *routines*)
The class for alcoholic residue.

__init__(*residue*, *optinstance*, *routines*)
Initialize the alcoholic class by removing the alcoholic hydrogen if it exists.

Parameters

- **residue** (Residue) – the residue to optimize
- **optinstance** (*OptimizationHandler*) – the optimization instance containing information about what to optimize
- **routines** (Debump) – debumping routines object

complete()
Complete an alcoholic optimization.

Call *finalize()* and then remove all extra LP atoms.

finalize()
Finalize an alcoholic residue.

Todo: Replace hard-coded values in the function.

Try to minimize conflict with nearby atoms by building away from them. Called when LPs are still present so as to account for their bonds.

try_acceptor(*acc*, *donor*)

Add a lone pair to an optimizeable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

try_both(*donor*, *acc*, *accobj*)

Attempt to optimize both the donor and acceptor.

If one is fixed, we only need to try one side. Otherwise first try to satisfy the donor - if that's succesful, try to satisfy the acceptor. An undo may be necessary if the donor is satisfied and the acceptor isn't.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor
- **accobj** (Flip) – a Flip-like hydrogen bond structure object

Returns indication of whether hydrogen was added

Return type bool

try_donor(*donor*, *acc*)

Add hydrogen to an optimizable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

class pdb2pqr.hydrogens.structures.**Carboxylic**(*residue*, *optinstance*, *routines*)

The class for carboxylic residues

__init__(*residue*, *optinstance*, *routines*)

Initialize carboxylic optimization class.

Initialize a case where the lone hydrogen atom can have four different orientations. Works similar to `initializeFlip()` by pre-adding the necessary atoms.

This also takes into account that the carboxyl group has different bond lengths for the two C-O bonds - this is probably due to one bond being assigned as a C=O. As a result hydrogens are only added to the C-O (longer) bond.

Parameters

- **residue** (Residue) – the residue to flip
- **optinstance** (*OptimizationHandler*) – the optimization instance containing information about what to optimize
- **routines** (Debump) – debumping routines object

complete()

If not already fixed, finalize the optimization.

finalize()
Finalize a protontated residue.

Try to minimize conflict with nearby atoms.

fix(*donor*, *acc*)
Fix the carboxylic residue.

is_carboxylic_hbond(*donor*, *acc*)
Determine whether this donor acceptor pair is a hydrogen bond.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor
- **accobj** (Flip) – a Flip-like hydrogen bond structure object

Returns indication of whether hydrogen was added

Return type bool

rename(*hydatom*)
Rename the optimized atoms appropriately.

This is done since the forcefields tend to require that the hydrogen is linked to a specific oxygen, and this atom may have different parameter values.

Parameters **hydatom** (Atom) – the hydrogen atom that was added

try_acceptor(*acc*, *donor*)
Add lone pair to an optimizable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

try_both(*donor*, *acc*, *accobj*)
Attempt to optimize donor and acceptor.

Called when both the donor and acceptor are optimizeable. If one is fixed, we only need to try one side. Otherwise first try to satisfy the donor - if that's succesful, try to satisfy the acceptor. An undo may be necessary if the donor is satisfied and the acceptor isn't.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor
- **accobj** (Flip) – a Flip-like hydrogen bond structure object

Returns indication of whether hydrogen was added

Return type bool

try_donor(*donor*, *acc*)
Add hydrogen to an optimizable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

class pdb2pqr.hydrogens.structures.**Flip**(*residue*, *optinstance*, *routines*)

The holder for optimization of flippable residues.

__init__(*residue*, *optinstance*, *routines*)

Initialize a potential flip.

Rather than flipping the given residue back and forth, take each atom that would be flipped and pre-flip it, making a new `*FLIP` atom in its place.

Parameters

- **residue** (Residue) – the residue to flip
- **optinstance** (*OptimizationHandler*) – the optimization instance containing information about what to optimize
- **routines** (Debump) – debumping routines object

complete()

Complete the flippable residue optimization.

Call the *finalize()* function, and then rename all `FLIP` atoms back to their standard names.

finalize()

Finalize a flippable group back to its original state.

Since the original atoms are now `*FLIP`, it deletes the `*` atoms and renames the `*FLIP` atoms back to `*`.

fix_flip(*bondatom*)

Remove atoms if hydrogen bond has been found for specified atom.

If bondatom is of type `*FLIP`, remove all `*` atoms, otherwise remove all `*FLIP` atoms.

Parameters **bondatom** (Atom) – atom to flip

try_acceptor(*acc*, *donor*)

Aa lone pair to an optimizable residue.

Parameters

- **acc** (Atom) – hydrogen bond acceptor
- **donor** (Atom) – hydrogen bond donor

Returns indication of whether addition was successful

Return type bool

try_both(*donor*, *acc*, *accobj*)

Try to optimize both donor and acceptor bonds.

Called when both the donor and acceptor are optimizeable. If one is fixed, we only need to try one side. Otherwise first try to satisfy the donor - if that's succesful, try to satisfy the acceptor. An undo may be necessary if the donor is satisfied and the acceptor isn't.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

- **accobj** (Flip) – a Flip-like hydrogen bond structure object

Returns indication of whether hydrogen was added

Return type bool

try_donor(*donor*, *acc*)
Add hydrogen to an optimizable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

class pdb2pqr.hydrogens.structures.**Generic**(*residue*, *optinstance*, *routines*)
Generic optimization class

__init__(*residue*, *optinstance*, *routines*)
Initialize a potential optimization.

Parameters

- **residue** (Residue) – the residue to optimize
- **optinstance** (*OptimizationHandler*) – the optimization instance containing information about what to optimize
- **routines** (Debump) – debumping routines object

complete()
If not already fixed, finalize.

finalize()
Initialize some variable and pass.

class pdb2pqr.hydrogens.structures.**HydrogenAmbiguity**(*residue*, *hdef*, *routines*)
Contains information about ambiguities in hydrogen conformations.

__init__(*residue*, *hdef*, *routines*)
If the residue has a rotateable hydrogen, remove it.

If it can be flipped, pre-flip the residue by creating all additional atoms.

Parameters

- **residue** (Residue) – the residue in question
- **hdef** – the hydrogen definition matching the residue
- **routines** (Debump) – the debumping routines object

class pdb2pqr.hydrogens.structures.**HydrogenConformation**(*hname*, *boundatom*, *bondlength*)
Class for possible hydrogen conformations.

The *HydrogenConformation* class contains data about possible hydrogen conformations as specified in the hydrogen data file.

__init__(*hname*, *boundatom*, *bondlength*)

Parameters

- **hname** – The hydrogen name (string)

- **boundatom** – The atom the hydrogen is bound to (string)
- **bondlength** – The bond length (float)

add_atom(*atom*)
Add an atom to the list of atoms.

Parameters **atom** (DefinitionAtom) – the atom to be added

class pdb2pqr.hydrogens.structures.**HydrogenDefinition**(*name*, *opttype*, *optangle*, *map_*)
Class for potential ambiguities in amino acid hydrogens.

It is essentially the hydrogen definition file in object form.

__init__(*name*, *opttype*, *optangle*, *map_*)
Initialize the object with information from the definition file.

See HYDROGENS.XML for more information.

Parameters

- **name** (*str*) – the name of the grouping
- **opttype** (*str*) – the optimization type of the grouping
- **optangle** (*str*) – the optimization angle of the grouping
- **map** – the map of hydrogens

add_conf(*conf*)
Add a hydrogen conformation to the list.

Parameters **conf** (HydrogenConformation) – the conformation to be added

class pdb2pqr.hydrogens.structures.**HydrogenHandler**
Extends the SAX XML Parser to parse the Hydrogens.xml class.

__init__()
Initialize self. See help(type(self)) for accurate signature.

characters(*text*)
Set a given attribute of the object to the text.

Parameters **text** (*str*) – value of the attribute

endElement(*name*)
Complete object is passed in by name parameter.

Todo: Rename this and related methods to conform with PEP8

Parameters **name** (*str*) – name for element

startElement(*name*, *_*)
Create optimization holder objects or atoms.

Todo: Rename this and related methods to conform with PEP8

Parameters **name** (*str*) – name for element

class pdb2pqr.hydrogens.structures.**PotentialBond**(*atom1*, *atom2*, *dist*)

A class containing the hydrogen bond structure.

__init__(*atom1*, *atom2*, *dist*)

Initialize the class.

Parameters

- **atom1** (Atom) – the first atom in the potential bond
- **atom2** (Atom) – the second atom in the potential bond
- **dist** (*float*) – the distance between the two atoms

class pdb2pqr.hydrogens.structures.**Water**(*residue*, *optinstance*, *routines*)

The class for water residues.

__init__(*residue*, *optinstance*, *routines*)

Initialize the water optimization class.

Parameters

- **residue** (Residue) – the residue to flip
- **optinstance** (*OptimizationHandler*) – the optimization instance containing information about what to optimize
- **routines** (Debump) – debumping routines object

complete()

Complete the water optimization process.

finalize()

Finalize a water residue.

Try to minimize conflict with nearby atoms by building away from them. Called when LPs are still present so as to account for their bonds.

try_acceptor(*acc*, *donor*)

The main driver for adding an LP to an optimizeable residue.

Todo: This looks like a bug: donorh ends up being the last item in donor.bonds. This may be fixed by setting a best_donorh to go with bestdist and using best_donorh in the function below

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

try_both(*donor*, *acc*, *accobj*)

Attempt to optimize both the donor and the acceptor.

If one is fixed, we only need to try one side. Otherwise first try to satisfy the donor - if that's successful, try to satisfy the acceptor. An undo may be necessary if the donor is satisfied and the acceptor isn't.

Parameters

- **donor** (Atom) – hydrogen bond donor

- **acc** (Atom) – hydrogen bond acceptor
- **accobj** (Flip) – a Flip-like hydrogen bond structure object

Returns indication of whether hydrogen was added

Return type bool

try_donor(*donor*, *acc*)
Add hydrogen to an optimizable residue.

Parameters

- **donor** (Atom) – hydrogen bond donor
- **acc** (Atom) – hydrogen bond acceptor

Returns indication of whether addition was successful

Return type bool

ligand and submodule contents

ligand

Ligand support functions.

Todo: Some of the definitions in this module belong in a configuration file other than here.

Code author: Jens Erik Nielsen

hydrogens.ligand.mol2

Support molecules in Tripos MOL2 format.

For further information look at (web page exists: 25 August 2005): http://www.tripos.com/index.php?family=modules,SimplePage,,,&page=sup_mol2&s=0

class pdb2pqr.ligand.mol2.**Mol2Atom**
MOL2 molecule atoms.

__init__()
Initialize self. See help(type(self)) for accurate signature.

assign_radius(*primary_dict*, *secondary_dict*)
Assign radius to atom.

Todo: It seems inconsistent that this function pulls radii from a dictionary and the biomolecule routines use force field files.

Parameters

- **primary_dict** (*dict*) – primary dictionary of radii indexed by atom type or element
- **secondary_dict** (*dict*) – backup dictionary for radii not found in primary dictionary

bond_order
Total number of electrons in bonds with other atoms.

Returns total number of electrons in bonds with other atoms

Return type int

bonded_atom_names
Bonded atom names.

Returns bonded atom names

Return type list

coords
Coordinates.

Returns coordinates

Return type numpy.ndarray

distance(*other*)
Get distance between two atoms.

Parameters **other** (`Mol2Atom`) – other atom for distance measurement

Returns distance

Return type float

element
Element for this atom (uppercase).

Returns element for this atom

Return type str

formal_charge
Formal charge for this atom

Returns formal charge for this atom

Return type int

num_bonded_heavy
Number of heavy atoms bonded to this atom.

Returns number of heavy atoms

Return type int

num_bonded_hydrogen
Number of hydrogen atoms bonded to this atom.

Returns number of hydrogen atoms

Return type int

class pdb2pqr.ligand.mol2.**Mol2Bond**(*atom1*, *atom2*, *bond_type*, *bond_id=0*)
MOL2 molecule bonds.

__init__(*atom1*, *atom2*, *bond_type*, *bond_id=0*)
Initialize bond.

Parameters

- **atom1** (*str*) – name of first atom in bond

- **atom2** (*str*) – name of second atom in bond
- **bond_type** (*int*) – type of bond: 1 (single), 2 (double), or ar (aromatic)
- **bond_id** (*int*) – integer ID of bond

atom_names

Get atom names in bond.

Returns tuple with names of atoms in bond.

Return type (str, str)

length

Get bond length.

Returns bond length

Return type float

class pdb2pqr.ligand.mol2.**Mol2Molecule**

Tripos MOL2 molecule.

__init__()

Initialize self. See help(type(self)) for accurate signature.

assign_charges()

Assign charges to atoms in molecule.

assign_parameters(*primary_dict={'C': 1.87, 'Cl': 1.82, 'F': 2.4, 'H': 1.1, 'I': 2.65, 'N': 1.4, 'O.co2': 1.76, 'S': 2.15}, secondary_dict={'Ar': 1.88, 'As': 1.85, 'Br': 1.85, 'C': 1.7, 'Cl': 1.75, 'F': 1.47, 'H': 1.2, 'He': 1.4, 'I': 1.98, 'Kr': 2.02, 'N': 1.55, 'Ne': 1.54, 'O': 1.52, 'P': 1.8, 'S': 1.8, 'Se': 1.9, 'Si': 2.1, 'Te': 2.06, 'Xe': 2.16}*)

Assign charges and radii to atoms in molecule.

Parameters

- **primary_dict** – primary dictionary of radii indexed by atom type or element
- **secondary_dict** – backup dictionary for radii not found in primary dictionary

assign_radii(*primary_dict, secondary_dict*)

Assign radii to atoms in molecule.

Parameters

- **primary_dict** (*dict*) – primary dictionary of radii indexed by atom type or element
- **secondary_dict** (*dict*) – backup dictionary for radii not found in primary dictionary

find_atom_torsions(*start_atom*)

Set the torsion angles that start with this atom (name).

Parameters **start_atom** (*str*) – starting atom name

Returns list of 4-tuples containing atom names comprising torsions

find_new_rings(*path, rings, level=0*)

Find new rings in molecule.

This was borrowed from StackOverflow: https://j.mp/2AHaukj

Parameters

- **path** (*list of str*) – list of atom names
- **rings** (*list of str*) – current list of rings

- **level** (*int*) – recursion level

Returns new list of rings

Return type int

parse_atoms(*mol2_file*)
Parse @<TRIPOS>ATOM section of file.

Parameters **mol2_file** – file-like object with MOL2 data

Returns file-like object advanced to bonds section

Raises

- **ValueError** – for bad MOL2 ATOM lines
- **TypeError** – for bad charge entries

parse_bonds(*mol2_file*)
Parse @<TRIPOS>BOND section of file.

Atoms must already have been parsed. Also sets up torsions and rings.

Parameters **mol2_file** – file-like object with MOL2 data

Returns file-like object advanced to SUBSTRUCTURE section

read(*mol2_file*)
Routines for reading MOL2 file.

Parameters **mol2_file** – file-like object with MOL2 data

static rotate_to_smallest(*path*)
Rotate cycle path so that it begins with the smallest node.

This was borrowed from StackOverflow: https://j.mp/2AHaukj

Parameters **path** (*list of str*) – list of atom names

Returns rotated path

Return type list of str

set_rings()
Set all rings in molecule.

This was borrowed from StackOverflow: https://j.mp/2AHaukj

set_torsions()
Set all torsions in molecule.

hydrogens.ligand.peoe

Implements the PEOE method.

The PEOE method is described in: Paul Czodrowski, Ingo Dramburg, Christoph A. Sotriffer Gerhard Klebe. Development, validation, and application of adapted PEOE charges to estimate pKa values of functional groups in protein–ligand complexes. Proteins, 65, 424-437, 2006. https://doi.org/10.1002/prot.21110

pdb2pqr.ligand.peoe.**assign_terms**(*atoms*, *term_dict*)
Assign polynomial terms to each atom.

Parameters

- **atoms** (*list*) – list of Mol2Atom atoms

- **term_dict** (*dict*) – dictionary of polynomial terms

Returns modified list of atoms

Return type list

`pdb2pqr.ligand.peoe.`**`electronegativity`**(*charge*, *poly_terms*, *atom_type*)

Calculate the electronegativity.

Calculation is based on a third-order polynomial in the atomic charge as described in Equation 2 of https://doi.org/10.1002/prot.21110.

Parameters

- **charge** (*float*) – charge of atom
- **poly_terms** – polynomial terms ordered from 0th- to 3rd-order
- **atom_type** (*str*) – string with atom type

Returns electronegativity value

Return type float

Raises **IndexError** – if incorrect number of poly_terms given

`pdb2pqr.ligand.peoe.`**`equilibrate`**(*atoms*, *damp=0.778*, *scale=1.56*, *num_cycles=6*, *term_dict={'BR': (10.88, 8.47, 1.16, 19.71), 'C.1': (10.39, 9.45, 0.73, 20.57), 'C.2': (9.29, 9.32, 1.51, 19.62), 'C.3': (7.98, 9.18, 1.88, 19.04), 'C.AR': (8.530000000000001, 9.18, 1.88, 19.04), 'C.CAT': (7.98, 9.18, 1.88, 19.04), 'CL': (10.38, 9.69, 1.35, 22.04), 'F': (12.36, 13.85, 2.31, 30.82), 'H': (7.17, 6.24, -0.56, 12.85), 'I': (10.9, 7.96, 0.96, 18.82), 'N.1': (15.68, 11.7, -0.27, 27.11), 'N.2': (12.87, 11.15, 0.85, 24.87), 'N.3': (17.54, 10.28, 1.36, 28.0), 'N.4': (17.54, 10.28, 1.36, 28.0), 'N.AM': (16.369999999999997, 11.15, 0.85, 24.87), 'N.AR': (11.579999999999998, 11.15, 0.85, 24.87), 'N.PL3': (13.37, 11.15, 0.85, 24.87), 'O.2': (14.18, 12.92, 1.39, 28.49), 'O.3': (11.08, 12.92, 1.39, 28.49), 'O.CO2': (15.25, 13.79, 0.47, 31.33), 'O.OH': (14.98, 12.92, 1.39, 28.49), 'P.3': (10.63, 9.13, 1.38, 20.65), 'S.2': (10.63, 9.13, 1.38, 20.65), 'S.3': (10.63, 9.13, 1.38, 20.65), 'S.O2': (10.63, 9.13, 1.38, 20.65)}*)

Equilibrate the atomic charges.

Parameters

- **atoms** (*list*) – list of Mol2Atom atoms to equilibrate
- **damp** (*float*) – damping factor for equilibration process
- **scale** (*float*) – scaling factor for equilibration process
- **num_cycles** (*int*) – number of PEOE cycles
- **term_dict** (*dict*) – dictionary of polynomial terms

Returns revised list of atoms

Return type list

`hydrogens.ligand.topology`

Ligand topology classes.

class pdb2pqr.ligand.topology.**Topology**(*molecule*)
Ligand topology class.

__init__(*molecule*)
Initialize with molecule.

Parameters **molecule** (Mol2Molecule) – Mol2Molecule object

na

Nucleic Acid Structures for PDB2PQR

This module contains the base nucleic acid structures for pdb2pqr.

Code author: Todd Dolinsky

Code author: Nathan Baker

class pdb2pqr.na.**ADE**(*atoms*, *ref*)
Adenosine class.

__init__(*atoms*, *ref*)
Initialize residue.

Parameters **atoms** (*[Atom]*) – add atoms to residue

letter_code()
Return letter code for nucleic acid.

Returns letter code for nucleic acid

Return type str

set_state()
Set ribo- vs. deoxyribo- state of this residue.

class pdb2pqr.na.**CYT**(*atoms*, *ref*)
Cytidine class

__init__(*atoms*, *ref*)
Initialize residue.

Parameters **atoms** (*[Atom]*) – add atoms to residue

letter_code()
Return letter code for nucleic acid.

Returns letter code for nucleic acid

Return type str

set_state()
Set ribo- vs. deoxyribo- state of this residue.

pdb2pqr.na.**DA**
alias of *pdb2pqr.na.ADE*

pdb2pqr.na.**DC**
alias of *pdb2pqr.na.CYT*

pdb2pqr.na.**DG**
alias of *pdb2pqr.na.GUA*

pdb2pqr.na.**DT**
alias of *pdb2pqr.na.THY*

class pdb2pqr.na.**GUA**(*atoms*, *ref*)

Guanosine class

__init__(*atoms*, *ref*)

Initialize residue.

Parameters **atoms** (*[Atom]*) – add atoms to residue

letter_code()

Return letter code for nucleic acid.

Returns letter code for nucleic acid

Return type str

set_state()

Set ribo- vs. deoxyribo- state of this residue.

class pdb2pqr.na.**Nucleic**(*atoms*, *ref*)

This class provides standard features of the nucleic acids listed below.

__init__(*atoms*, *ref*)

Initialize the class

Parameters **atoms** (*list*) – list of atom-like (HETATM or ATOM) objects to be stored

add_atom(*atom*)

Add existing atom to system.

Override the existing add_atom - include the link to the reference object.

Parameters **atom** (Atom) – atom to add to system.

add_dihedral_angle(*value*)

Add the value to the list of chi angles.

Parameters **value** (*float*) – dihedral angle to add to list (in degrees)

create_atom(*atomname*, *newcoords*)

Create an atom.

Overrides the generic residue's create_atom().

Todo: This code is duplicated in several places.

Parameters

- **atomname** (*str*) – the name of the atom to add
- **newcoords** (*[(float, float, float)]*) – the coordinates of the atom

set_state()

Adds the termini for all inherited objects.

pdb2pqr.na.**RA**

alias of *pdb2pqr.na.URA*

pdb2pqr.na.**RC**

alias of *pdb2pqr.na.CYT*

pdb2pqr.na.**RG**

alias of *pdb2pqr.na.GUA*

class pdb2pqr.na.**THY**(*atoms*, *ref*)
Thymine class

__init__(*atoms*, *ref*)
Initialize residue.

Parameters **atoms** (*[Atom]*) – add atoms to residue

letter_code()
Return letter code for nucleic acid.

Returns letter code for nucleic acid

Return type str

set_state()
Set ribo- vs. deoxyribo- state of this residue.

class pdb2pqr.na.**URA**(*atoms*, *ref*)
Uridine class

__init__(*atoms*, *ref*)
Initialize residue.

Parameters **atoms** (*[Atom]*) – add atoms to residue

letter_code()
Return letter code for nucleic acid.

Returns letter code for nucleic acid

Return type str

set_state()
Set ribo- vs. deoxyribo- state of this residue.

biomolecule

The biomolecule object used in PDB2PQR and associated methods.

Todo: This module should be broken into separate files.

Authors: Todd Dolinsky, Yong Huang

class pdb2pqr.biomolecule.**Biomolecule**(*pdblist*, *definition*)
Biomolecule class.

This class represents the parsed PDB, and provides a hierarchy of information - each Biomolecule object contains a list of Chain objects as provided in the PDB file. Each Chain then contains its associated list of Residue objects, and each Residue contains a list of Atom objects, completing the hierarchy.

__init__(*pdblist*, *definition*)
Initialize using parsed PDB file

Parameters

- **pdblist** (*list*) – list of objects from pdb from lines of PDB file
- **definition** (Definition) – topology definition object

add_hydrogens()

Add the hydrogens to the biomolecule.

This requires either the rebuild_tetrahedral function for tetrahedral geometries or the standard quatfit methods. These methods use three nearby bonds to rebuild the atom; the closer the bonds, the more accurate the results. As such the peptide bonds are used when available.

apply_force_field(*forcefield_*)

Apply the forcefield to the atoms within the biomolecule.

Parameters **forcefield** (`Forcefield`) – forcefield object

Returns (list of atoms that were found in the forcefield, list of atoms that were not found in the forcefield)

Return type (list, list)

apply_name_scheme(*forcefield_*)

Apply the naming scheme of the given forcefield.

Parameters **forcefield** (`Forcefield`) – forcefield object

apply_patch(*patchname*, *residue*)

Apply a patch to the given residue.

This is one of the key functions in PDB2PQR. A similar function appears in `definitions` - that version is needed for residue level subtitutions so certain protonation states (i.e. CYM, HSE) are detectatble on input.

This version looks up the particular patch name in the patch_map stored in the biomolecule, and then applies the various commands to the reference and actual residue structures.

See the inline comments for a more detailed explanation.

Parameters

- **patchname** (*str*) – the name of the patch
- **residue** (`Residue`) – the residue to apply the patch to

apply_pka_values(*force_field*, *ph*, *pkadic*)

Apply calculated pKa values to assign titration states.

Parameters

- **force_field** (*str*) – force field name (determines naming of residues)
- **ph** (*float*) – pH value
- **pkadic** (*dict*) – dictionary of pKa values for residues

assign_termini(*chain*, *neutraln=False*, *neutralc=False*)

Assign the termini for the given chain.

Assignment made by looking at the start and end residues.

Parameters

- **chain** (`Chain`) – chain of biomolecule
- **neutraln** (*bool*) – indicate whether to neutralize N-terminus
- **neutralc** (*bool*) – indicate whether to neutralize C-terminus

atoms

Return all Atom objects in list format.

Returns all atom objects

Return type [*Atom*]

calculate_dihedral_angles()
Calculate dihedral angles for every residue in the biomolecule.

charge
Get the total charge on the biomolecule

Todo: Since the misslist is used to identify incorrect charge assignments, this routine does not list the 3 and 5 termini of nucleic acid chains as having non-integer charge even though they are (correctly) non-integer.

Returns (list of residues with non-integer charges, the total charge on the biomolecule)

Return type (list, float)

create_html_typemap(*definition*, *outfilename*)
Create an HTML typemap file at the desired location.

If a type cannot be found for an atom a blank is listed.

Parameters

- **definition** (`Definition`) – the definition objects.
- **outfilename** (*str*) – the name of the file to write

create_residue(*residue*, *resname*)
Create a residue object.

If the resname is a known residue type, try to make that specific object, otherwise just make a standard residue object.

Parameters

- **residue** (*list*) – a list of atoms
- **resname** (*str*) – the name of the residue

Returns the residue object

Return type *Residue*

hold_residues(*hlist*)
Set fixed state of specified residues.

Parameters **hlist** (*[(str, str, str)]*) – list of (res_seq, chainid, ins_code) specifying the residues for altering fixed state status.

num_bio_atoms
Return the number of ATOM (not HETATM) records in the biomolecule.

Returns number of ATOM records

Return type int

num_heavy
Return number of biomolecular heavy atoms in structure.

Todo: Figure out if this is redundant with *Biomolecule.num_bio_atoms()*

Note: Includes hydrogens (but those are stripped off eventually)

Returns number of heavy atoms

Return type int

num_missing_heavy
Return number of missing biomolecular heavy atoms in structure.

Returns number of missing heavy atoms in structure

Return type int

patch_map
Return definition patch maps.

Returns definition patch maps

Return type list

reference_map
Return definition reference map.

Returns definition reference map

Return type dict

remove_hydrogens()
Remove hydrogens from the biomolecule.

repair_heavy()
Repair all heavy atoms.

Unfortunately the first time we get to an atom we might not be able to rebuild it - it might depend on other atoms to be rebuild first (think side chains). As such a 'seenmap' is used to keep track of what we've already seen and subsequent attempts to rebuild the atom.

Raises **ValueError** – missing atoms prevent reconstruction

reserialize()
Generate new serial numbers for atoms in the biomolecule.

set_donors_acceptors()
Set the donors and acceptors within the biomolecule.

set_hip()
Set all HIS states to HIP.

set_reference_distance()
Set the distance to the CA atom in the residue.

This is necessary for determining which atoms are allowed to move during rotations. Uses the `shortest_path()` algorithm found in `utilities`.

Raises **ValueError** – if shortest path cannot be found (e.g., if the atoms are not connected)

set_states()

Set the state of each residue.

This is the last step before assigning the forcefield, but is necessary so as to distinguish between various protonation states.

See `aa` for residue-specific functions.

set_termini(*neutraln=False*, *neutralc=False*)

Set the termini for a protein.

First set all known termini by looking at the ends of the chain. Then examine each residue, looking for internal chain breaks.

Todo: This function needs to be cleaned and simplified

Parameters

- **neutraln** (*bool*) – indicate whether N-terminus is neutral
- **neutralc** (*bool*) – indicate whether C-terminus is neutral

update_bonds()

Update the bonding network of the biomolecule.

This happens in 3 steps:

1. Apply the PEPTIDE patch to all Amino residues to add reference for the N(i+1) and C(i-1) atoms
2. UpdateInternal_bonds for inter-residue linking
3. Set the links to the N(i+1) and C(i-1) atoms

update_internal_bonds()

Update the internal bonding network.

Update using the reference objects in each atom.

update_residue_types()

Find the type of residue as notated in the Amino Acid definition.

Todo: Why are we setting residue types to numeric values (see code)?

update_ss_bridges()

Check and set SS-bridge partners.

residue

Biomolecular residue class.

Code author: Todd Dolinsky

Code author: Nathan Baker

class pdb2pqr.residue.**Residue**(*atoms*)

Residue class

Todo: Should this class have a member variable for dihedrals? Pylint complains!

The residue class contains a list of Atom objects associated with that residue and other helper functions.

`__init__`(*atoms*)
Initialize the class

Parameters **`atoms`** (*`list`*) – list of atom-like (`HETATM` or `ATOM`) objects to be stored

`add_atom`(*atom*)
Add the atom object to the residue.

Parameters **`atom`** – atom-like object, e.g., `HETATM` or `ATOM`

`charge`
Get the total charge of the residue.

In order to get rid of floating point rounding error, do a string transformation.

Returns The charge of the residue (float)

Return type charge

`get_atom`(*name*)
Retrieve a residue atom based on its name.

Parameters **`resname`** (*`str`*) – name of the residue to retrieve

Returns residue

Return type *structures.Atom*

`get_moveable_names`(*pivot*)
Return all atom names that are further away than the pivot atom.

Parameters

- **`residue`** (`Residue`) – the residue to use
- **`pivot`** (*`str`*) – the pivot atomname

Returns names of atoms further away than pivot atom

Return type [str]

`has_atom`(*name*)
Return True if atom in residue.

Parameters **`name`** (*`str`*) – atom name in question

Returns True if atom in residue

Return type bool

`static letter_code`()
Default letter code for residue.

Returns letter code for residue

Return type str

`pick_dihedral_angle`(*conflict_names*, *oldnum=None*)
Choose an angle number to use in debumping.

Instead of simply picking a random chiangle, this function uses a more intelligent method to improve efficiency. The algorithm uses the names of the conflicting atoms within the residue to determine which angle number has the best chance of fixing the problem(s). The method also insures that the same chiangle will not be run twice in a row.

Parameters

- **`residue`** (`Residue`) – residue that is being debumped
- **`conflict_names`** (*`[str]`*) – list of atom names that are currently conflicts
- **`oldnum`** (*`int`*) – old dihedral angle number

Returns new dihedral angle number

Return type int

`remove_atom`(*atomname*)
Remove an atom from the residue object.

Parameters **`atomname`** (*`str`*) – the name of the atom to be removed

`rename_atom`(*oldname*, *newname*)
Rename an atom to a new name.

Parameters

- **`oldname`** (*`str`*) – old atom name
- **`newname`** (*`str`*) – new atom name

`rename_residue`(*name*)
Rename the residue.

Parameters **`name`** (*`str`*) – the new name of the residue

`reorder`()
Reorder the atoms to start with N, CA, C, O if they exist.

`classmethod rotate_tetrahedral`(*atom1*, *atom2*, *angle*)
Rotate about the atom1-atom2 bond by a given angle.

All atoms connected to atom2 will rotate.

Parameters

- **`atom1`** (`structures.Atom`) – first atom of the bond to rotate about
- **`atom2`** (`structures.Atom`) – second atom of the bond to rotate about
- **`angle`** (*`float`*) – degrees to rotate

`set_chain_id`(*value*)
Set the chain ID.

Parameters **`value`** (*`str`*) – new **`chain_id`** value

`set_donors_acceptors`()
Set the donors and acceptors within the residue.

`set_res_seq`(*value*)
Change the residue sequence number.

Set the atom field **`res_seq`** and change the residue's information. The **`icode`** field is no longer useful.

Parameters **`value`** (*`int`*) – the new value of **`res_seq`**

`update_terminus_status`()
Update the **`is_n_terms`** and **`is_c_term`** flags.

structures

Simple biomolecular structures.

This module contains the simpler structure objects used in PDB2PQR and their associated methods.

Code author: Todd Dolinsky

Code author: Nathan Baker

class `pdb2pqr.structures.`**`Atom`**(*atom=None*, *type_='ATOM'*, *residue=None*)

Represent an atom.

The Atom class inherits from the `ATOM` object in `pdb`. This class used for adding fields not found in the PDB that may be useful for analysis. This class also simplifies code by combining `ATOM` and `HETATM` objects into a single class.

`__init__`(*atom=None*, *type_='ATOM'*, *residue=None*)

Initialize the new Atom object by using the old object.

Parameters

- **atom** (`ATOM`) – the original ATOM object (could be None)
- **type** (*`str`*) – either ATOM or HETATM
- **residue** (`Residue`) – a pointer back to the parent residue object (could be None)

`add_bond`(*bondedatom*)

Add a bond to the list of bonds.

Parameters **bondedatom** (`ATOM`) – the atom to bond to

`coords`

Return the x,y,z coordinates of the atom.

Returns list of the coordinates

Return type [float, float, float]

classmethod **`from_pqr_line`**(*line*)

Create an atom from a PQR line.

Parameters

- **cls** (`Atom`) – class for classmethod
- **line** (*`str`*) – PQR line

Returns new atom or None (for REMARK and similar lines)

Return type *Atom*

Raises **`ValueError`** – for problems parsing

classmethod **`from_qcd_line`**(*line*, *atom_serial*)

Create an atom from a QCD (UHBD QCARD format) line.

Parameters

- **cls** (`Atom`) – class for classmethod
- **line** (*`str`*) – PQR line
- **atom_serial** (*`int`*) – atom serial number

Returns new atom or None (for REMARK and similar lines)

Return type *Atom*

Raises `ValueError` – for problems parsing

`get_common_string_rep`(*chainflag=False*)
Returns a string of the common column of the new atom type.

Uses the `ATOM` string output but changes the first field to either be `ATOM` or `HETATM` as necessary. This is used to create the output for PQR and PDB files.

Returns string with ATOM/HETATM field set appropriately

Return type str

`get_pdb_string`()
Returns a string of the atom type.

Uses the `ATOM` string output but changes the first field to either be `ATOM` or `HETATM` as necessary. This is for the PDB representation of the atom. The `propka` module depends on this being correct.

Returns string with ATOM/HETATM field set appropriately

Return type str

`get_pqr_string`(*chainflag=False*)
Returns a string of the atom type.

Uses the `ATOM` string output but changes the first field to either be `ATOM` or `HETATM` as necessary. This is used to create the output for PQR files.

Returns string with ATOM/HETATM field set appropriately

Return type str

`has_reference`
Determine if the object has a reference object or not.

All known atoms should have reference objects.

Returns whether atom has reference object

Return type bool

`is_backbone`
Return True if atom name is in backbone, otherwise False.

Returns whether atom is in backbone

Return type bool

`is_hydrogen`
Is this atom a Hydrogen atom?

Returns whether this atom is a hydrogen

Return type bool

`class` `pdb2pqr.structures.`**`Chain`**(*chain_id*)
Chain class

The chain class contains information about each chain within a given `Biomolecule` object.

`__init__`(*chain_id*)
Initialize the class.

Parameters **`chain_id`** (*`str`*) – ID for this chain as denoted in the PDB

add_residue(*residue*)

Add a residue to the chain

Parameters **residue** (Residue) – residue to be added

atoms

Return a list of Atom objects contained in this chain.

Returns list of Atom objects

Return type [*Atom*]

renumber_residues()

Renumber atoms.

Renumber based on actual residue number and not PDB **res_seq**

topology

Parser for topology XML files.

Code author: Nathan Baker

Code author: Yong Huang

class pdb2pqr.topology.**Topology**(*topology_file*)

Contains the structured definitions of residue reference coordinates as well as alternate titration, conformer, and tautomer states.

__init__(*topology_file*)

Initialize object.

Parameters **topology_file** (*file-like object*) – topology file object ready for reading

class pdb2pqr.topology.**TopologyAtom**(*parent*)

A class for atom topology information.

__init__(*parent*)

Initialize with an upper-level class that contains an atom array.

For example, initialize with *TopologyReference* or TopologyConformerAddition

Parameters **parent** – parent object with atom array

class pdb2pqr.topology.**TopologyConformer**(*topology_tautomer*)

A class for topology conformer information.

__init__(*topology_tautomer*)

Initialize object.

Parameters **topology_tautomer** (TopologyTautomer) – tautomer for which to evaluate conformers

class pdb2pqr.topology.**TopologyConformerAdd**(*topology_conformer*)

A class for adding atoms to a conformer.

__init__(*topology_conformer*)

Initialize object.

Parameters **topology_conformer** (TopologyConformer) – conformer to which to add atoms

class pdb2pqr.topology.**TopologyConformerRemove**(*topology_conformer*)
A class for removing atoms from a conformer.

__init__(*topology_conformer*)
Initialize object.

Parameters **topology_conformer** (TopologyConformer) – conformer to which to add atoms

class pdb2pqr.topology.**TopologyDihedral**(*parent*)
A class for dihedral angle topology information.

__init__(*parent*)
Initialize with a parent that has a dihedral list.

Parameters **parent** – parent object with a dihedral list

class pdb2pqr.topology.**TopologyHandler**
Handler for XML-based topology files.

Assumes the following hierarchy of tags:

- topology
  - residue
    - reference
    - titrationstate
      - tautomer
      - conformer

__init__()
Initialize self. See help(type(self)) for accurate signature.

characters(*text*)
Parse characters in topology XML file.

Parameters **text** (*str*) – XML character data

endElement(*tag_name*)
End parsing element.

Parameters **tag_name** (*str*) – name of XML element tag for which to end parsing

startElement(*tag_name*, *_*)
Start element parsing.

Parameters **tag_name** (*str*) – name of XML element tag to start parsing

class pdb2pqr.topology.**TopologyReference**(*topology_residue*)
A class for the reference structure of a residue.

__init__(*topology_residue*)
Initialize object.

Parameters **topology_residue** (TopologyResidue) – reference residue

class pdb2pqr.topology.**TopologyResidue**(*topology_*)
A class for residue topology information.

__init__(*topology_*)
Initialize with a Topology object.

Parameters **topology** – topology object

class pdb2pqr.topology.**TopologyTautomer**(*topology_titration_state*)
A class for topology tautomer information.

__init__(*topology_titration_state*)
Initialize object.

Parameters **topology_titration_state** (TopologyTitrationState) – titration state of this residue

class pdb2pqr.topology.**TopologyTitrationState**(*topology_residue*)
A class for the titration state of a residue.

__init__(*topology_residue*)
Initialize object.

Parameters **topology_residue** (TopologyResidue) – residue topology

2.7.3 I/O modules

cif

CIF parsing methods.

This methods use the pdbx/cif parser provided by WWPDB (http://mmcif.wwpdb.org/docs/sw-examples/python/html/index.html)

Code author: Juan Brandi

pdb2pqr.cif.**atom_site**(*block*)
Handle ATOM_SITE block.

Data items in the ATOM_SITE category record details about the atom sites in a macromolecular crystal structure, such as the positional coordinates, atomic displacement parameters, magnetic moments and directions. (Source: https://j.mp/2Zprx41)

Parameters **block** (*[str]*) – PDBx data block

Returns (array of pdb.ATOM objects, array of things that weren't handled by parser)

Return type ([*Atom*], [str])

pdb2pqr.cif.**author**(*block*)
Handle AUTHOR block.

Parameters **block** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

pdb2pqr.cif.**cispep**(*block*)
Handle CISPEP block.

Parameters **block** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

pdb2pqr.cif.**compnd**(*block*)
Handle COMPND block.

Parameters **block** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`conect`**(*block*)

Handle CONECT block.

Data items in the STRUCT_CONN category record details about the connections between portions of the structure. These can be hydrogen bonds, salt bridges, disulfide bridges and so on.

The `STRUCT_CONN_TYPE` records define the criteria used to identify these connections. (Source: https://j.mp/3gPkJT5)

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`count_models`**(*block*)

Count models in structure file block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns number of models in block

Return type int

`pdb2pqr.cif.`**`cryst1`**(*block*)

Handle CRYST1 block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`expdata`**(*block*)

Handle EXPDTA block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`header`**(*block*)

Handle HEADER block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`keywds`**(*block*)

Handle KEYWDS block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`origxn`**(*block*)

Handle ORIGXn block.

Parameters **`block`** (*[str]*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`read_cif`**`(`*`cif_file`*`)`

Parse CIF-format data into array of Atom objects.

Todo: Manage several blocks of data.

Parameters **`file`** (*`file`*) – open file-like object

Returns (a dictionary indexed by PDBx/CIF record names, a list of record names that couldn't be parsed)

Return type (dict, [str])

`pdb2pqr.cif.`**`scalen`**`(`*`block`*`)`

Handle SCALEn block.

Parameters **`block`** (*`[str]`*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`source`**`(`*`block`*`)`

Handle SOURCE block.

Parameters **`block`** (*`[str]`*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`ssbond`**`(`*`block`*`)`

Handle SSBOND block.

Parameters **`block`** (*`[str]`*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`pdb2pqr.cif.`**`title`**`(`*`block`*`)`

Handle TITLE block.

Parameters **`block`** (*`[str]`*) – PDBx data block

Returns (array of pdb.conect objects, array of things that did not parse)

Return type ([*pdb.CONECT*], [str])

`io`

Functions related to reading and writing data.

`class` `pdb2pqr.io.`**`DuplicateFilter`**

Filter duplicate messages.

`__init__``()`

Initialize a filter.

Initialize with the name of the logger which, together with its children, will have its events allowed through the filter. If no name is specified, allow every event.

filter(*record*)
Filter current record.

pdb2pqr.io.**dump_apbs**(*output_pqr*, *output_path*)
Generate and dump APBS input files related to output_pqr.

Parameters

- **output_pqr** (*str*) – path to PQR file used to generate APBS input file
- **output_path** (*str*) – path for APBS input file output

pdb2pqr.io.**get_definitions**(*aa_path='AA.xml'*, *na_path='NA.xml'*, *patch_path='PATCHES.xml'*)
Load topology definition files.

Parameters

- **aa_path** (*str*) – likely location of amino acid topology definitions
- **na_path** (*str*) – likely location of nucleic acid topology definitions
- **patch_path** (*str*) – likely location of patch topology definitions

Returns topology Definitions object.

Return type *Definition*

pdb2pqr.io.**get_molecule**(*input_path*)
Get molecular structure information as a series of parsed lines.

Parameters **input_path** – structure file PDB ID or path

Returns (list of molecule records, Boolean indicating whether entry is CIF)

Return type ([str], bool)

Raises **RuntimeError** – problems with structure file

pdb2pqr.io.**get_old_header**(*pdblist*)
Get old header from list of pdb objects.

Parameters **pdblist** (*[]*) – list of pdb block objects

Returns old header as string

Return type str

pdb2pqr.io.**get_pdb_file**(*name*)
Obtain a PDB file.

First check the path given on the command line - if that file is not available, obtain the file from the PDB webserver at http://www.rcsb.org/pdb/

Todo: This should be a context manager (to close the open file).

Todo: Remove hard-coded parameters.

Parameters **name** (*str*) – name of PDB file (path) or PDB ID

Returns file-like object containing PDB file

Return type file

`pdb2pqr.io.`**`print_biomolecule_atoms`**(*atomlist*, *chainflag=False*, *pdbfile=False*)

Get PDB-format text lines for specified atoms.

Parameters

- **`atomlist`** (*`[Atom]`*) – the list of atoms to include
- **`chainflag`** (*`bool`*) – flag whether to print chainid or not

Returns list of strings, each representing an atom PDB line

Return type [str]

`pdb2pqr.io.`**`print_pqr_header`**(*pdblist*, *atomlist*, *reslist*, *charge*, *force_field*, *ph_calc_method*, *ph*, *ffout*, *include_old_header=False*)

Print the header for the PQR file.

Parameters

- **`pdblist`** (*`[str]`*) – list of lines from original PDB with header
- **`atomlist`** (*`[Atom]`*) – a list of atoms that were unable to have charges assigned
- **`reslist`** (*`[Residue]`*) – a list of residues with non-integral charges
- **`charge`** (*`float`*) – the total charge on the biomolecule
- **`force_field`** (*`str`*) – the forcefield name
- **`ph_calc_method`** (*`str`*) – pKa calculation method
- **`ph`** (*`float`*) – pH value, if any
- **`ffout`** (*`str`*) – forcefield used for naming scheme

Returns the header for the PQR file

Return type str

`pdb2pqr.io.`**`print_pqr_header_cif`**(*atomlist*, *reslist*, *charge*, *force_field*, *ph_calc_method*, *ph*, *ffout*, *include_old_header=False*)

Print the header for the PQR file in CIF format.

Parameters

- **`atomlist`** (*`[Atom]`*) – a list of atoms that were unable to have charges assigned
- **`reslist`** (*`[Residue]`*) – a list of residues with non-integral charges
- **`charge`** (*`float`*) – the total charge on the biomolecule
- **`force_field`** (*`str`*) – the forcefield name
- **`ph_calc_method`** (*`str`*) – pKa calculation method
- **`ph`** (*`float`*) – pH value, if any
- **`ffout`** (*`str`*) – forcefield used for naming scheme

Returns the header for the PQR file

Return type str

`pdb2pqr.io.`**`read_dx`**(*dx_file*)

Read DX-format volumetric information.

The OpenDX file format is defined at <https://www.idvbook.com/wp-content/uploads/2010/12/opendx.pdf‘.

Note: This function is not a general-format OpenDX file parser and makes many assumptions about the input data type, grid structure, etc.

Todo: This function should be moved into the APBS code base.

Parameters **`dx_file`** (*`file`*) – file object for DX file, ready for reading as text

Returns dictionary with data from DX file

Return type dict

Raises **`ValueError`** – on parsing error

`pdb2pqr.io.`**`read_pqr`**(*pqr_file*)

Read PQR file.

Parameters **`pqr_file`** (*`file`*) – file object ready for reading as text

Returns list of atoms read from file

Return type [*Atom*]

`pdb2pqr.io.`**`read_qcd`**(*qcd_file*)

Read QCD (UHDB QCARD format) file.

Parameters **`qcd_file`** (*`file`*) – file object ready for reading as text

Returns list of atoms read from file

Return type [*Atom*]

`pdb2pqr.io.`**`setup_logger`**(*output_pqr*, *level='DEBUG'*)

Setup the logger.

Setup logger to output the log file to the same directory as PQR output.

Parameters

- **`output_pqr`** (*`str`*) – path to PQR file
- **`level`** (*`str`*) – logging level

`pdb2pqr.io.`**`test_dat_file`**(*name*)

Test for the forcefield file with a few name permutations.

Parameters **`name`** (*`str`*) – the name of the dat file

Returns the path to the file

Return type Path

Raises **`FileNotFoundError`** – file not found

`pdb2pqr.io.`**`test_for_file`**(*name*, *type_*)

Test for the existence of a file with a few name permutations.

Parameters

- **`name`** (*`str`*) – name of file
- **`type`** (*`str`*) – type of file

Returns path to file

Raises `FileNotFoundError` – if file not found

Return type Path

`pdb2pqr.io.`**`test_names_file`**`(`*name*`)`

Test for the .names file that contains the XML mapping.

Parameters **`name`** (*`str`*) – the name of the forcefield

Returns the path to the file

Return type Path

Raises `FileNotFoundError` – file not found

`pdb2pqr.io.`**`test_xml_file`**`(`*name*`)`

Test for the XML file with a few name permutations.

Parameters **`name`** (*`str`*) – the name of the dat file

Returns the path to the file

Return type Path

Raises `FileNotFoundError` – file not found

`pdb2pqr.io.`**`write_cube`**`(`*cube_file*, *data_dict*, *atom_list*, *comment='CPMD CUBE FILE.'*`)`

Write a Cube-format data file.

Cube file format is defined at <https://docs.chemaxon.com/display/Gaussian_Cube_format.html>.

Todo: This function should be moved into the APBS code base.

Parameters

- **`cube_file`** (*`file`*) – file object ready for writing text data
- **`data_dict`** (*`dict`*) – dictionary of volumetric data as produced by *`read_dx()`*
- **`comment`** (*`str`*) – comment for Cube file

`inputgen`

Create an APBS input file using `psize` data.

Code author: Todd Dolinsky

Code author: Nathan Baker

`class` `pdb2pqr.inputgen.`**`Elec`**`(`*pqrpath*, *size*, *method*, *asyncflag*, *istrng=0*, *potdx=False*`)`

An object for the ELEC section of an APBS input file.

`__init__``(`*pqrpath*, *size*, *method*, *asyncflag*, *istrng=0*, *potdx=False*`)`

Initialize object.

Todo: Remove hard-coded parameters.

Parameters

- **`pqrpath`** (*`str`*) – path to PQR file

- **size** (Psize) – parameter sizing object
- **method** (*str*) – solution method (e.g., mg-para, mg-auto, etc.)
- **asyncflag** (*bool*) – perform an asynchronous parallel focusing calculation
- **istrng** – ionic strength/concentration (M)
- **potdx** (*bool*) – whether to write out potential information in DX format

class pdb2pqr.inputgen.**Input**(*pqrpath*, *size*, *method*, *asyncflag*, *istrng=0*, *potdx=False*)

Each object of this class is one APBS input file.

__init__(*pqrpath*, *size*, *method*, *asyncflag*, *istrng=0*, *potdx=False*)

Initialize the input file class.

Each input file contains a PQR name, a list of elec objects, and a list of strings containing print statements. For starters, assume two ELEC statements are needed, one for the inhomgenous and the other for the homogenous dielectric calculations.

Note: This assumes you have already run psize, either by `size.run_psize(...)()` or `size.parse_string(...)()` followed by `size.set_all()`.

Parameters

- **pqrpath** (*str*) – path to PQR file
- **size** (Psize) – parameter sizing object
- **method** (*str*) – solution method (e.g., mg-para, mg-auto, etc.)
- **asyncflag** (*bool*) – perform an asynchronous parallel focusing calculation
- **istrng** – ionic strength/concentration (M)
- **potdx** (*bool*) – whether to write out potential information in DX format

dump_pickle()

Make a Python pickle associated with the APBS input parameters.

Todo: is this function still useful?

print_input_files(*output_path*)

Generate the input file(s) associated with this object.

Parameters **output_path** (*str*) – location for generated files

pdb2pqr.inputgen.**build_parser**()

Build argument parser.

pdb2pqr.inputgen.**main**()

Main driver

pdb2pqr.inputgen.**split_input**(*filename*)

Split the parallel input file into multiple async file names.

Parameters **filename** (*str*) – the path to the original parallel input file

pdb

PDB parsing class

This module parses PDBs in accordance to PDB Format Description Version 2.2 (1996); it is not very forgiving. Each class in this module corresponds to a record in the PDB Format Description. Much of the documentation for the classes is taken directly from the above PDB Format Description.

Code author: Todd Dolinsky

Code author: Yong Huang

Code author: Nathan Baker

class pdb2pqr.pdb.**ANISOU**(*line*)

ANISOU class

The ANISOU records present the anisotropic temperature factors.

__init__(*line*)

Initialize by parsing line:

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number.
13-16	string	name	Atom name.
17	string	alt_loc	Alternate location indicator.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Insertion code.
29-35	int	u00	U(1,1)
36-42	int	u11	U(2,2)
43-49	int	u22	U(3,3)
50-56	int	u01	U(1,2)
57-63	int	u02	U(1,3)
64-70	int	u12	U(2,3)
73-76	string	seg_id	Segment identifier, left-justified.
77-78	string	element	Element symbol, right-justified.
79-80	string	charge	Charge on the atom.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**ATOM**(*line*)

ATOM class

The ATOM records present the atomic coordinates for standard residues. They also present the occupancy and temperature factor for each atom. Heterogen coordinates use the HETATM record type. The element symbol is always present on each ATOM record; segment identifier and charge are optional.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number.
13-16	string	name	Atom name.
17	string	alt_loc	Alternate location indicator.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Code for insertion of residues.
31-38	float	x	Orthogonal coordinates for X in Angstroms.
39-46	float	y	Orthogonal coordinates for Y in Angstroms.
47-54	float	z	Orthogonal coordinates for Z in Angstroms.
55-60	float	occupancy	Occupancy.
61-66	float	temp_factor	Temperature factor.
73-76	string	seg_id	Segment identifier, left-justified.
77-78	string	element	Element symbol, right-justified.
79-80	string	charge	Charge on the atom.

Parameters **`line`** (`str`) – line with PDB class

class `pdb2pqr.pdb.`**`AUTHOR`**(*line*)

AUTHOR field

The AUTHOR record contains the names of the people responsible for the contents of the entry.

`__init__`(*line*)

Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	author_list	List of the author names, separated by commas

Parameters **`line`** (`str`) – line with PDB class

class `pdb2pqr.pdb.`**`BaseRecord`**(*line*)

Base class for all records.

Verifies the received record type.

`__init__`(*line*)

Initialize self. See help(type(self)) for accurate signature.

`record_type`()

Return PDB record type as string.

Returns record type

Return type str

class `pdb2pqr.pdb.`**`CAVEAT`**(*line*)

CAVEAT field

CAVEAT warns of severe errors in an entry. Use caution when using an entry containing this record.

`__init__`(*line*)

Initialize by parsing line.

COLUMNS	TYPE	FIELD	DEFINITION
12-15	string	id_code	PDB ID code of this entry.
20-70	string	comment	Free text giving the reason for the CAVEAT.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**CISPEP**(*line*)

CISPEP field

CISPEP records specify the prolines and other peptides found to be in the cis conformation. This record replaces the use of footnote records to list cis peptides.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	ser_num	Record serial number.
12-14	string	pep1	Residue name.
16	string	chain_id1	Chain identifier.
18-21	int	seq_num1	Residue sequence number.
22	string	icode1	Insertion code.
26-28	string	pep2	Residue name.
30	string	chain_id2	Chain identifier.
32-35	int	seq_num2	Residue sequence number.
36	string	icode2	Insertion code.
44-46	int	mod_num	Identifies the specific model.
54-59	float	measure	Measure of the angle in degrees.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**COMPND**(*line*)

COMPND field

The COMPND record describes the macromolecular contents of an entry. Each macromolecule found in the entry is described by a set of token: value pairs, and is referred to as a COMPND record component. Since the concept of a molecule is difficult to specify exactly, PDB staff may exercise editorial judgment in consultation with depositors in assigning these names.

For each macromolecular component, the molecule name, synonyms, number assigned by the Enzyme Commission (EC), and other relevant details are specified.

__init__(*line*)

Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	compound	Description of the molecular list components.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**CONECT**(*line*)

CONECT class

The CONECT records specify connectivity between atoms for which coordinates are supplied. The connectivity is described using the atom serial number as found in the entry. CONECT records are mandatory for HET groups (excluding water) and for other bonds not specified in the standard residue connectivity table which involve atoms in standard residues (see Appendix 4 for the list of standard residues). These records are generated by the PDB.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number
12-16	int	serial1	Serial number of bonded atom
17-21	int	serial2	Serial number of bonded atom
22-26	int	serial3	Serial number of bonded atom
27-31	int	serial4	Serial number of bonded atom
32-36	int	serial5	Serial number of hydrogen bonded atom
37-41	int	serial6	Serial number of hydrogen bonded atom
42-46	int	serial7	Serial number of salt bridged atom
47-51	int	serial8	Serial number of hydrogen bonded atom
52-56	int	serial9	Serial number of hydrogen bonded atom
57-61	int	serial10	Serial number of salt bridged atom

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`CRYST1`**(*line*)
CRYST1 class

The CRYST1 record presents the unit cell parameters, space group, and Z value. If the structure was not determined by crystallographic means, CRYST1 simply defines a unit cube.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
7-15	float	a	a (Angstroms).
16-24	float	b	b (Angstroms).
25-33	float	c	c (Angstroms).
34-40	float	alpha	alpha (degrees).
41-47	float	beta	beta (degrees).
48-54	float	gamma	gamma (degrees).
56-66	string	space_group	Space group.
67-70	int	z	Z value.

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`DBREF`**(*line*)
DBREF field

The DBREF record provides cross-reference links between PDB sequences and the corresponding database entry or entries. A cross reference to the sequence database is mandatory for each peptide chain with a length greater than ten (10) residues. For nucleic acid entries a DBREF record pointing to the Nucleic Acid Database (NDB) is mandatory when the corresponding entry exists in NDB.

`__init__`(*line*)
Initialize by parsing a line.

COLUMNS	TYPE	FIELD	DEFINITION
8-11	string	id_code	ID code of this entry.
13	string	chain_id	Chain identifier.
15-18	int	seq_begin	Initial sequence number of the PDB sequence segment.
19	string	in-sert_begin	Initial insertion code of the PDB sequence segment.
21-24	int	seq_end	Ending sequence number of the PDB sequence segment.
25	string	in-sert_end	Ending insertion code of the PDB sequence segment.
27-32	string	database	Sequence database name. "PDB" when a corresponding sequence database entry has not been identified.
34-41	string	db_accession	Sequence database accession code. For GenBank entries, this is the NCBI gi number.
43-54	string	db_id_code	Sequence database identification code. For GenBank entries, this is the accession code.
56-60	int	db_seq_begin	Initial sequence number of the database seqment.
61	string	db_ins_begin	Insertion code of initial residue of the segment, if PDB is the reference.
63-67	int	db-seq_end	Ending sequence number of the database segment.
68	string	db_ins_end	Insertion code of the ending residue of the segment, if PDB is the reference.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**END**(*line*)

END class

The END records are paired with MODEL records to group individual structures found in a coordinate entry.

__init__(*line*)

Initialize with line.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**ENDMDL**(*line*)

ENDMDL class

The ENDMDL records are paired with MODEL records to group individual structures found in a coordinate entry.

__init__(*line*)

Initialize self. See help(type(self)) for accurate signature.

class pdb2pqr.pdb.**EXPDTA**(*line*)

EXPDTA field

The EXPDTA record identifies the experimental technique used. This may refer to the type of radiation and sample, or include the spectroscopic or modeling technique. Permitted values include:

- ELECTRON DIFFRACTION
- FIBER DIFFRACTION
- FLUORESCENCE TRANSFER
- NEUTRON DIFFRACTION
- NMR

- THEORETICAL MODEL
- X-RAY DIFFRACTION

__init__(*line*)
Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	technique	The experimental technique(s) with optional comment describing the sample or experiment

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**FORMUL**(*line*)
FORMUL field

The FORMUL record presents the chemical formula and charge of a non-standard group.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
9-10	int	comp_num	Component number
13-15	string	hetatm_id	Het identifier
19	string	asterisk *	for water
20-70	string	text	Chemical formula

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**HEADER**(*line*)
HEADER field

The HEADER record uniquely identifies a PDB entry through the id_code field. This record also provides a classification for the entry. Finally, it contains the date the coordinates were deposited at the PDB.

__init__(*line*)
Initialize by parsing a line.

COLUMNS	TYPE	FIELD	DEFINITION
11-50	string	classification	Classifies the molecule(s)
51-59	string	dep_date	Deposition date. This is the date the coordinates were received by the PDB
63-66	string	id_code	This identifier is unique wihin within PDB

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**HELIX**(*line*)
HELIX field

HELIX records are used to identify the position of helices in the molecule. Helices are both named and numbered. The residues where the helix begins and ends are noted, as well as the total length.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	ser_num	Serial number of the helix. This starts at 1 and increases incrementally.
12-14	string	helix_id	Helix identifier. In addition to a serial number, each helix is given an alphanumeric character helix identifier.
16-18	string	init_res_name	Name of the initial residue.
20	string	init_chain_id	Chain identifier for the chain containing this helix.
22-25	int	init_seq_num	Sequence number of the initial residue.
26	string	init_i_code	Insertion code of the initial residue.
28-30	string	end_res_name	Name of the terminal residue of the helix.
32	string	end_chain_id	Chain identifier for the chain containing this helix.
34-37	int	end_seq_num	Sequence number of the terminal residue.
38	string	end_i_code	Insertion code of the terminal residue.
39-40	int	he-lix_class	Helix class (see below).
41-70	string	comment	Comment about this helix.
72-76	int	length	Length of this helix.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**HET**(*line*)

HET field

HET records are used to describe non-standard residues, such as prosthetic groups, inhibitors, solvent molecules, and ions for which coordinates are supplied. Groups are considered HET if they are:

- not one of the standard amino acids, and
- not one of the nucleic acids (C, G, A, T, U, and I), and
- not one of the modified versions of nucleic acids (+C, +G, +A, +T, +U, and +I), and
- not an unknown amino acid or nucleic acid where UNK is used to indicate the unknown residue name.

Het records also describe heterogens for which the chemical identity is unknown, in which case the group is assigned the hetatm_id UNK.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	string	hetatm_id	Het identifier, right-justified.
13	string	ChainID	Chain identifier.
14-17	int	seq_num	Sequence number.
18	string	ins_code	Insertion code.
21-25	int	num_het_atoms	Number of HETATM records.
31-70	string	text	Text describing Het group.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**HETATM**(*line*, *sybyl_type='A.aaa'*, *l_bonds=[]*, *l_bonded_atoms=[]*)

HETATM class

The HETATM records present the atomic coordinate records for atoms within “non-standard” groups. These records are used for water molecules and atoms presented in HET groups.

__init__(*line*, *sybyl_type='A.aaa'*, *l_bonds=[]*, *l_bonded_atoms=[]*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number.
13-16	string	name	Atom name.
17	string	alt_loc	Alternate location indicator.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Code for insertion of residues.
31-38	float	x	Orthogonal coordinates for X in Angstroms.
39-46	float	y	Orthogonal coordinates for Y in Angstroms.
47-54	float	z	Orthogonal coordinates for Z in Angstroms.
55-60	float	occupancy	Occupancy.
61-66	float	temp_factor	Temperature factor.
73-76	string	seg_id	Segment identifier, left- justified.
77-78	string	element	Element symbol, right-justified.
79-80	string	charge	Charge on the atom.

Parameters **line** (`str`) – line with PDB class

class pdb2pqr.pdb.**HETNAM**(*line*)
HETNAM field

This record gives the chemical name of the compound with the given hetatm_id.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
12-14	string	hetatm_id	Het identifier, right-justified.
16-70	string	text	Chemical name.

Parameters **line** (`str`) – line with PDB class

class pdb2pqr.pdb.**HETSYN**(*line*)
HETSYN field

This record provides synonyms, if any, for the compound in the corresponding (i.e., same hetatm_id) HETNAM record. This is to allow greater flexibility in searching for HET groups.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
12-14	string	hetatm_id	Het identifier, right-justified.
16-70	string	hetatm_synonyms	List of synonyms

Parameters **line** (`str`) – line with PDB class

class pdb2pqr.pdb.**HYDBND**(*line*)
HYDBND field

The HYDBND records specify hydrogen bonds in the entry.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
13-16	string	name1	Atom name.
17	string	alt_loc1	Alternate location indicator.
18-20	string	res_name1	Residue name.
22	string	chain1	Chain identifier.
23-27	int	res_seq1	Residue sequence number.
28	string	i_code1	Insertion code.
30-33	string	name_h	Hydrogen atom name.
34	string	alt_loc_h	Alternate location indicator.
36	string	chain_h	Chain identifier.
37-41	int	res_seq_h	Residue sequence number.
42	string	ins_codeH	Insertion code.
44-47	string	name2	Atom name.
48	string	alt_loc2	Alternate location indicator.
49-51	string	res_name2	Residue name.
53	string	chain_id2	Chain identifier.
54-58	int	res_seq2	Residue sequence number.
59	string	ins_code2	Insertion code.
60-65	string	sym1	Symmetry operator for 1st non-hydrogen atom.
67-72	string	sym2	Symmetry operator for 2nd non-hydrogen atom.

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`JRNL`**(*line*)
JRNL field

The JRNL record contains the primary literature citation that describes the experiment which resulted in the deposited coordinate set. There is at most one JRNL reference per entry. If there is no primary reference, then there is no JRNL reference. Other references are given in REMARK 1.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
13-70	string	text	See details on web.

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`KEYWDS`**(*line*)
KEYWDS field

The KEYWDS record contains a set of terms relevant to the entry. Terms in the KEYWDS record provide a simple means of categorizing entries and may be used to generate index files. This record addresses some of the limitations found in the classification field of the HEADER record. It provides the opportunity to add further annotation to the entry in a concise and computer-searchable fashion.

`__init__`(*line*)
Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	keywds	Comma-separated list of keywords relevant to the entry

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`LINK`**(*line*)

LINK field

The LINK records specify connectivity between residues that is not implied by the primary structure. Connectivity is expressed in terms of the atom names. This record supplements information given in CONECT records and is provided here for convenience in searching.

`__init__`(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
13-16	string	name1	Atom name.
17	string	alt_loc1	Alternate location indicator.
18-20	string	res_name1	Residue name.
22	string	chain_id1	Chain identifier.
23-26	int	res_seq1	Residue sequence number.
27	string	ins_code1	Insertion code.
43-46	string	name2	Atom name.
47	string	alt_loc2	Alternate location indicator.
48-50	string	res_name2	Residue name.
52	string	chain_id2	Chain identifier.
53-56	int	res_seq2	Residue sequence number.
57	string	ins_code2	Insertion code.
60-65	string	sym1	Symmetry operator for 1st atom.
67-72	string	sym2	Symmetry operator for 2nd atom.

Parameters **`line`** (*`str`*) – line with PDB class

class `pdb2pqr.pdb.`**`MASTER`**(*line*)

MASTER class

The MASTER record is a control record for bookkeeping. It lists the number of lines in the coordinate entry or file for selected record types.

`__init__`(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
11-15	int	num_remark	Number of REMARK records
21-25	int	num_het	Number of HET records
26-30	int	numHelix	Number of HELIX records
31-35	int	numSheet	Number of SHEET records
36-40	int	numTurn	Number of TURN records
41-45	int	numSite	Number of SITE records
46-50	int	numX-form	Number of coordinate transformation records (ORIGX+SCALE+MTRIX)
51-55	int	numCoord	Number of atomic coordinate records (ATOM+HETATM)
56-60	int	numTer	Number of TER records
61-65	int	num-Conect	Number of CONECT records
66-70	int	numSeq	Number of SEQRES records

Parameters **`line`** (*str*) – line with PDB class

class pdb2pqr.pdb.**MODEL**(*line*)

MODEL class

The MODEL record specifies the model serial number when multiple structures are presented in a single coordinate entry, as is often the case with structures determined by NMR.

`__init__`(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
11-14	int	serial	Model serial number.

Parameters **`line`** (*str*) – line with PDB class

class pdb2pqr.pdb.**MODRES**(*line*)

MODRES field

The MODRES record provides descriptions of modifications (e.g., chemical or post-translational) to protein and nucleic acid residues. Included are a mapping between residue names given in a PDB entry and standard residues.

`__init__`(*line*)

Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
8-11	string	id_code	ID code of this entry.
13-15	string	res_name	Residue name used in this entry.
17	string	chain_id	Chain identifier.
19-22	int	seq_num	Sequence number.
23	string	ins_code	Insertion code.
25-27	string	stdRes	Standard residue name.
30-70	string	comment	Description of the residue modification.

Parameters **`line`** (*str*) – line with PDB class

class pdb2pqr.pdb.**MTRIX1**(*line*)
MATRIX1 PDB entry

class pdb2pqr.pdb.**MTRIX2**(*line*)
MATRIX2 PDB entry

class pdb2pqr.pdb.**MTRIX3**(*line*)
MATRIX3 PDB entry

class pdb2pqr.pdb.**MTRIXn**(*line*)
MTRIXn baseclass

The MTRIXn (n = 1, 2, or 3) records present transformations expressing non-crystallographic symmetry.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	serial	Serial number
11-20	float	mn1	M31
21-30	float	mn2	M32
31-40	float	mn3	M33
46-55	float	vn	V3
60	int	i_given	1 if coordinates for the representations which are approximately related by the transformations of the molecule are contained in the entry. Otherwise, blank.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**NUMMDL**(*line*)
NUMMDL class

The NUMMDL record indicates total number of models in a PDB entry.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
11-14	int	modelNumber	Number of models.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**OBSLTE**(*line*)
OBSLTE field

This record acts as a flag in an entry which has been withdrawn from the PDB's full release. It indicates which, if any, new entries have replaced the withdrawn entry.

The format allows for the case of multiple new entries replacing one existing entry.

__init__(*line*)
Initialize by parsing a line.

COLUMNS	TYPE	FIELD	DEFINITION
12-20	string	replace_date	Date that this entry was replaced.
22-25	string	id_code	ID code of this entry.
32-35	string	rid_code	ID code of entry that replaced this one.
37-40	string	rid_code	ID code of entry that replaced this one.
42-45	string	rid_code	ID code of entry that replaced this one.
47-50	string	rid_code	ID code of entry that replaced this one.
52-55	string	rid_code	ID code of entry that replaced this one.
57-60	string	rid_code	ID code of entry that replaced this one.
62-65	string	rid_code	ID code of entry that replaced this one.
67-70	string	rid_code	ID code of entry that replaced this one.

Parameters **`line`** (*`str`*) – line with PDB class

`class` `pdb2pqr.pdb.`**`ORIGX1`**(*line*)
ORIGX3 PDB entry

`class` `pdb2pqr.pdb.`**`ORIGX2`**(*line*)
ORIGX2 PDB entry

`class` `pdb2pqr.pdb.`**`ORIGX3`**(*line*)
ORIGX3 PDB entry

`class` `pdb2pqr.pdb.`**`ORIGXn`**(*line*)
ORIGXn class

The ORIGXn (n = 1, 2, or 3) records present the transformation from the orthogonal coordinates contained in the entry to the submitted coordinates.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
11-20	float	on1	O21
21-30	float	on2	O22
31-40	float	on3	O23
46-55	float	tn	T2

Parameters **`line`** (*`str`*) – line with PDB class

`class` `pdb2pqr.pdb.`**`REMARK`**(*line*)
REMARK field

REMARK records present experimental details, annotations, comments, and information not included in other records. In a number of cases, REMARKs are used to expand the contents of other record types. A new level of structure is being used for some REMARK records. This is expected to facilitate searching and will assist in the conversion to a relational database.

`__init__`(*line*)
Initialize by parsing line.

Parameters **`line`** (*`str`*) – line with PDB class

`class` `pdb2pqr.pdb.`**`REVDAT`**(*line*)
REVDAT field

REVDAT records contain a history of the modifications made to an entry since its release.

__init__(*line*)
Initialize by parsing a line.

Todo: If multiple modifications are present, only the last one in the file is preserved.

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	mod_num	Modification number.
14-22	string	mod_date	Date of modification (or release for new entries).
24-28	string	mod_id	Identifies this particular modification. It links to the archive used internally by PDB.
32	int	mod_type	An integer identifying the type of modification. In case of revisions with more than one possible mod_type, the highest value applicable will be assigned.
40-45	string	record	Name of the modified record.
47-52	string	record	Name of the modified record.
54-59	string	record	Name of the modified record.
61-66	string	record	Name of the modified record.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SCALE1**(*line*)
SCALE2 PDB entry

class pdb2pqr.pdb.**SCALE2**(*line*)
SCALE2 PDB entry

class pdb2pqr.pdb.**SCALE3**(*line*)
SCALE3 PDB entry

class pdb2pqr.pdb.**SCALEn**(*line*)
SCALEn baseclass

The SCALEn (n = 1, 2, or 3) records present the transformation from the orthogonal coordinates as contained in the entry to fractional crystallographic coordinates. Non-standard coordinate systems should be explained in the remarks.

__init__(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
11-20	float	sn1	S31
21-30	float	sn2	S32
31-40	float	sn3	S33
46-55	float	un	U3

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SEQADV**(*line*)
SEQADV field

The SEQADV record identifies conflicts between sequence information in the ATOM records of the PDB entry and the sequence database entry given on DBREF. Please note that these records were designed to identify differences and not errors. No assumption is made as to which database contains the correct data. PDB may include REMARK records in the entry that reflect the depositor's view of which database has the correct sequence.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-11	string	id_code	ID code of this entry.
13-15	string	res_name	Name of the PDB residue in conflict.
17	string	chain_id	PDB chain identifier.
19-22	int	seq_num	PDB sequence number.
23	string	ins_code	PDB insertion code.
25-28	string	database	Sequence database name.
30-38	string	db_id_code	Sequence database accession number.
40-42	string	db_res	Sequence database residue name.
44-48	int	db_seq	Sequence database sequence number.
50-70	string	conflict	Conflict comment.

Parameters **`line`** (`str`) – line with PDB class

class `pdb2pqr.pdb.`**`SEQRES`**(*line*)
SEQRES field

SEQRES records contain the amino acid or nucleic acid sequence of residues in each chain of the macromolecule that was studied.

`__init__`(*line*)
Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
9-10	int	ser_num	Serial number of the SEQRES record for the current chain. Starts at 1 and increments by one each line. Reset to 1 for each chain.
12	string	chain_id	Chain identifier. This may be any single legal character, including a blank which is used if there is only one chain.
14-17	int	num_res	Number of residues in the chain. This value is repeated on every record.
20-22	string	res_name	Residue name.
24-26	string	res_name	Residue name.
28-30	string	res_name	Residue name.
32-34	string	res_name	Residue name.
36-38	string	res_name	Residue name.
40-42	string	res_name	Residue name.
44-46	string	res_name	Residue name.
48-50	string	res_name	Residue name.
52-54	string	res_name	Residue name.
56-58	string	res_name	Residue name.
60-62	string	res_name	Residue name.
64-66	string	res_name	Residue name.
68-70	string	res_name	Residue name.

Parameters **`line`** (`str`) – line with PDB class

class `pdb2pqr.pdb.`**`SHEET`**(*line*)
SHEET field

SHEET records are used to identify the position of sheets in the molecule. Sheets are both named and numbered. The residues where the sheet begins and ends are noted.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	strand	Strand number which starts at 1 for each strand within a sheet and increases by one.
12-14	string	sheet_id	Sheet identifier.
15-16	int	num_strands	Number of strands in sheet.
18-20	string	init_res_name	Residue name of initial residue.
22	string	init_chain_id	Chain identifier of initial residue in strand.
23-26	int	init_seq_num	Sequence number of initial residue in strand.
27	string	init_i_code	Insertion code of initial residue in strand.
29-31	string	end_res_name	Residue name of terminal residue.
33	string	end_chain_id	Chain identifier of terminal residue.
34-37	int	end_seq_num	Sequence number of terminal residue.
38	string	end_i_code	Insertion code of terminal residue.
39-40	int	sense	Sense of strand with respect to previous strand in the sheet. 0 if first strand, 1 if parallel, -1 if anti-parallel.
42-45	string	cur_atom	Registration. Atom name in current strand.
46-48	string	curr_res_name	Registration. Residue name in current strand.
50	string	cur-ChainId	Registration. Chain identifier in current strand.
51-54	int	curr_res_seq	Registration. Residue sequence number in current strand.
55	string	curr_ins_code	Registration. Insertion code in current strand.
57-60	string	prev_atom	Registration. Atom name in previous strand.
61-63	string	prev_res_name	Registration. Residue name in previous strand.
65	string	pre-vChainId	Registration. Chain identifier in previous strand.
66-69	int	prev_res_seq	Registration. Residue sequence number in previous strand.
70	string	prev_ins_code	Registration. Insertion code in previous strand.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SIGATM**(*line*)

SIGATM class

The SIGATM records present the standard deviation of atomic parameters as they appear in ATOM and HET-ATM records.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number.
13-16	string	name	Atom name.
17	string	alt_loc	Alternate location indicator.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Code for insertion of residues.
31-38	float	sig_x	Standard deviation of orthogonal coordinates for X in Angstroms.
39-46	float	sig_y	Standard deviation of orthogonal coordinates for Y in Angstroms.
47-54	float	sig_z	Standard deviation of orthogonal coordinates for Z in Angstroms.
55-60	float	sig_occ	Standard deviation of occupancy.
61-66	float	sig_temp	Standard deviation of temperature factor.
73-76	string	seg_id	Segment identifier, left-justified.
77-78	string	element	Element symbol, right-justified.
79-80	string	charge	Charge on the atom.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SIGUIJ**(*line*)

SIGUIJ class

The SIGUIJ records present the anisotropic temperature factors.

__init__(*line*)

Initialize by parsing line:

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Atom serial number.
13-16	string	name	Atom name.
17	string	alt_loc	Alternate location indicator.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Insertion code.
29-35	int	sig11	Sigma U(1,1)
36-42	int	sig22	Sigma U(2,2)
43-49	int	sig33	Sigma U(3,3)
50-56	int	sig12	Sigma U(1,2)
57-63	int	sig13	Sigma U(1,3)
64-70	int	sig23	Sigma U(2,3)
73-76	string	seg_id	Segment identifier, left-justified.
77-78	string	el.ment	Element symbol, right-justified.
79-80	string	charge	Charge on the atom.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SITE**(*line*)

SITE class

The SITE records supply the identification of groups comprising important sites in the macromolecule.

`__init__`(*line*)
Initialize by parsing the line

COLUMNS 8-10	TYPE int	FIELD seq_num	DEFINITION Sequence number.
12-14	string	site_id	Site name.
16-17	int	num_res	Number of residues comprising site.
19-21	string	res_name1	Residue name for first residue comprising site.
23	string	chain_id1	Chain identifier for first residue comprising site.
24-27	int	seq1	Residue sequence number for first residue comprising site.
28	string	ins_code1	Insertion code for first residue comprising site.
30-32	string	res_name2	Residue name for second residue comprising site.
34	string	chain_id2	Chain identifier for second residue comprising site.
35-38	int	seq2	Residue sequence number for second residue comprising site.
39	string	ins_code2	Insertion code for second residue comprising site.
41-43	string	res_name3	Residue name for third residue comprising site.
45	string	chain_id3	Chain identifier for third residue comprising site.
46-49	int	seq3	Residue sequence number for third residue comprising site.
50	string	ins_code3	Insertion code for third residue comprising site.
52-54	string	res_name4	Residue name for fourth residue comprising site.
56	string	chain_id4	Chain identifier for fourth residue comprising site.
57-60	int	seq4	Residue sequence number for fourth residue comprising site.
61	string	ins_code4	Insertion code for fourth residue comprising site.

Parameters **`line`** (*`str`*) – line with PDB class

`class` `pdb2pqr.pdb.`**`SLTBRG`**(*line*)
SLTBRG field

The SLTBRG records specify salt bridges in the entry. records and is provided here for convenience in searching.

`__init__`(*line*)
Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
13-16	string	name1	Atom name.
17	string	alt_loc1	Alternate location indicator.
18-20	string	res_name1	Residue name.
22	string	chain_id1	Chain identifier.
23-26	int	res_seq1	Residue sequence number.
27	string	ins_code1	Insertion code.
43-46	string	name2	Atom name.
47	string	alt_loc2	Alternate location indicator.
48-50	string	res_name2	Residue name.
52	string	chain_id2	Chain identifier.
53-56	int	res_seq2	Residue sequence number.
57	string	ins_code2	Insertion code.
60-65	string	sym1	Symmetry operator for 1st atom.
67-72	string	sym2	Symmetry operator for 2nd atom.

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SOURCE**(*line*)

SOURCE field

The SOURCE record specifies the biological and/or chemical source of each biological molecule in the entry. Sources are described by both the common name and the scientific name, e.g., genus and species. Strain and/or cell-line for immortalized cells are given when they help to uniquely identify the biological entity studied.

__init__(*line*)

Initialize by parsing a line

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	source	Identifies the source of the macromolecule in a token: value format

Parameters **line** (*str*) – line with PDB class

class pdb2pqr.pdb.**SPRSDE**(*line*)

SPRSDE field

The SPRSDE records contain a list of the ID codes of entries that were made obsolete by the given coordinate entry and withdrawn from the PDB release set. One entry may replace many. It is PDB policy that only the principal investigator of a structure has the authority to withdraw it.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
12-20	string	super_date	Date this entry superseded the listed entries.
22-25	string	id_code	ID code of this entry.
32-35	string	sid_code	ID code of a superseded entry.
37-40	string	sid_code	ID code of a superseded entry.
42-45	string	sid_code	ID code of a superseded entry.
47-50	string	sid_code	ID code of a superseded entry.
52-55	string	sid_code	ID code of a superseded entry.
57-60	string	sid_code	ID code of a superseded entry.
62-65	string	sid_code	ID code of a superseded entry.
67-70	string	sid_code	ID code of a superseded entry.

Parameters **`line`** (*str*) – line with PDB class

class `pdb2pqr.pdb.`**`SSBOND`**(*line*)

SSBOND field

The SSBOND record identifies each disulfide bond in protein and polypeptide structures by identifying the two residues involved in the bond.

`__init__`(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	ser_num	Serial number.
16	string	chain_id1	Chain identifier.
18-21	int	seq_num1	Residue sequence number.
22	string	icode1	Insertion code.
30	string	chain_id2	Chain identifier.
32-35	int	seq_num2	Residue sequence number.
36	string	icode2	Insertion code.
60-65	string	sym1	Symmetry operator for 1st residue.
67-72	string	sym2	Symmetry operator for 2nd residue.

Parameters **`line`** (*str*) – line with PDB class

class `pdb2pqr.pdb.`**`TER`**(*line*)

TER class

The TER record indicates the end of a list of ATOM/HETATM records for a chain.

`__init__`(*line*)

Initialize by parsing line:

COLUMNS	TYPE	FIELD	DEFINITION
7-11	int	serial	Serial number.
18-20	string	res_name	Residue name.
22	string	chain_id	Chain identifier.
23-26	int	res_seq	Residue sequence number.
27	string	ins_code	Insertion code.

Parameters **`line`** (*str*) – line with PDB class

class pdb2pqr.pdb.**TITLE**(*line*)

TITLE field

The TITLE record contains a title for the experiment or analysis that is represented in the entry. It should identify an entry in the PDB in the same way that a title identifies a paper.

__init__(*line*)

Initialize by parsing a line.

COLUMNS	TYPE	FIELD	DEFINITION
11-70	string	title	Title of the experiment

Parameters line (*str*) – line with PDB class

class pdb2pqr.pdb.**TURN**(*line*)

TURN field

The TURN records identify turns and other short loop turns which normally connect other secondary structure segments.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	seq	Turn number; starts with 1 and increments by one.
12-14	string	turn_id	Turn identifier.
16-18	string	init_res_name	Residue name of initial residue in turn.
20	string	init_chain_id	Chain identifier for the chain containing this turn.
21-24	int	init_seq_num	Sequence number of initial residue in turn.
25	string	init_i_code	Insertion code of initial residue in turn.
27-29	string	end_res_name	Residue name of terminal residue of turn.
31	string	end_chain_id	Chain identifier for the chain containing this turn.
32-35	int	end_seq_num	Sequence number of terminal residue of turn.
36	string	end_i_code	Insertion code of terminal residue of turn.
41-70	string	comment	Associated comment.

Parameters line (*str*) – line with PDB class

class pdb2pqr.pdb.**TVECT**(*line*)

TVECT class

The TVECT records present the translation vector for infinite covalently connected structures.

__init__(*line*)

Initialize by parsing line

COLUMNS	TYPE	FIELD	DEFINITION
8-10	int	serial	Serial number
11-20	float	t1	Components of translation vector
21-30	float	t2	Components of translation vector
31-40	float	t2	Components of translation vector
41-70	string	text	Comments

Parameters line (*str*) – line with PDB class `pdb2pqr.pdb.`**`read_atom`**`(`*line*`)`

If the ATOM/HETATM is not column-formatted, try to get some information by parsing whitespace from the right. Look for five floating point numbers followed by the residue number.

Parameters **`line`** (`str`) – the line to parse

`pdb2pqr.pdb.`**`read_pdb`**`(`*file_*`)`

Parse PDB-format data into array of Atom objects.

Parameters **`file`** (`file`) – open File-like object

Returns (a list of objects from this module, a list of record names that couldn't be parsed)

Return type (list, list)

`pdb2pqr.pdb.`**`register_line_parser`**`(`*klass*`)`

Register a line parser in the global dictionary.

Parameters **`klass`** – class for line parser

2.7.4 Other modules

`cells`

Cell list to facilitate neighbor searching.

`class` `pdb2pqr.cells.`**`Cells`**`(`*cellsize*`)`

Accelerate the search for nearby atoms.

A pure all versus all search is O(n^2) - for every atom, every other atom must be searched. This is rather inefficient, especially for large biomolecules where cells may be tens of angstroms apart. The cell class breaks down the xyz biomolecule space into several 3-D cells of desired size - then by simply examining atoms that fall into the adjacent cells one can quickly find nearby cells.

`__init__``(`*cellsize*`)`

Initialize the cells.

Parameters **`cellsize`** (`int`) – the size of each cell (in Angstroms)

`add_cell``(`*atom*`)`

Add an atom to the cell.

Parameters **`atom`** (`Atom`) – the atom to add

`assign_cells``(`*biomolecule*`)`

Place each atom in a virtual cell for easy neighbor comparison.

Parameters **`biomolecule`** (`Biomolecule`) – biomolecule with atoms to assign to cells

`get_near_cells``(`*atom*`)`

Find all atoms in cells bordering an atom.

Parameters **`atom`** (`Atom`) – the atom to test

Returns a list of nearby atoms

Return type [*Atom*]

`remove_cell``(`*atom*`)`

Remove an atom from a cell.

Parameters **`atom`** (`Atom`) – the atom to remove

debump

Routines for biomolecule optimization.

Code author: Jens Erik Nielsen

Code author: Todd Dolinsky

Code author: Yong Huang

Code author: Nathan Baker

class pdb2pqr.debump.**Debump**(*biomolecule*, *definition=None*)

Grab bag of random stuff that apparently didn't fit elsewhere.

Todo: This class needs to be susbtantially refactored in to multiple classes with clear responsibilities.

__init__(*biomolecule*, *definition=None*)

Initialize the Debump object.

Parameters

- **biomolecule** (Biomolecule) – the biomolecule to debump
- **definition** (Definition) – topology definition file

debump_biomolecule()

Minimize bump score for molecule.

Make sure that none of the added atoms were rebuilt on top of existing atoms. See each called function for more information.

Raises **ValueError** – if missing (backbone) atoms are encountered

debump_residue(*residue*, *conflict_names*)

Debump a specific residue.

Only should be called if the residue has been detected to have a conflict. If called, try to rotate about dihedral angles to resolve the conflict.

Parameters

- **residue** (Residue) – the residue in question
- **conflict_names** (*[str]*) – a list of atomnames that were rebuilt too close to other atoms

Returns True if successful, False otherwise

Return type bool

find_nearby_atoms(*atom*)

Find nearby atoms for conflict-checking.

Uses neighboring cells to compare atoms rather than an all versus all O(n^2) algorithm, which saves a great deal of time. There are several instances where we ignore potential conflicts; these include donor/acceptor pairs, atoms in the same residue, and bonded CYS bridges.

Parameters **atom** (Atom) – find nearby atoms to this atom

Returns a dictionary of `Atom too close` to `amount of overlap for that atom`

Return type dict

find_residue_conflicts(*residue*, *write_conflict_info=False*)
Find conflicts between residues.

Parameters

- **residue** (Residue) – residue to check
- **write_conflict_info** (*bool*) – write verbose output about conflict

Returns list of conflicts

Return type [str]

get_bump_score(*residue*)
Get a bump score for a residue.

Parameters **residue** (Residue) – residue with bumping to evaluate

Returns bump score

Return type float

get_bump_score_atom(*atom*)
Find nearby atoms for conflict-checking.

Uses neighboring cells to compare atoms rather than an all versus all O(n^2) algorithm, which saves a great deal of time. There are several instances where we ignore potential conflicts; these include donor/acceptor pairs, atoms in the same residue, and bonded CYS bridges.

Parameters **atom** (Atom) – find nearby atoms to this atom

Returns a bump score sum((dist-cutoff)**20 for all nearby atoms

Return type float

get_closest_atom(*atom*)
Get the closest atom that does not form a donor/acceptor pair.

Used to detect potential conflicts.

Note: Cells must be set before using this function.

Parameters **atom** (Atom) – the atom to test

Returns the closest atom to the input atom that does not satisfy a donor/acceptor pair.

Return type *Atom*

score_dihedral_angle(*residue*, *anglenum*)
Assign score to dihedral angle.

Parameters

- **residue** (Residue) – residue with dihedral angles to score
- **anglenum** (*int*) – specific dihedral angle index

Returns score for dihedral angle

Return type float

set_dihedral_angle(*residue*, *anglenum*, *angle*)
Rotate a residue about a given angle.

Uses `quatfit` methods to perform the matrix mathematics.

Parameters

- **`residue`** (`Residue`) – residue to rotate
- **`anglenum`** – specific dihedral angle index
- **`angle`** (*`float`*) – the angle to set the dihedral to

`main`

Perform functions related to _main_ execution of PDB2PQR.

This module is intended for functions that directly touch arguments provided at the invocation of PDB2PQR. It was created to avoid cluttering the __init__.py file.

Code author: Nathan Baker (et al.)

`pdb2pqr.main.`**`build_main_parser`**`()`

Build an argument parser.

Todo: Need separate argparse groups for PDB2PKA and PROPKA. These exist but need real options.

Returns argument parser

Return type argparse.ArgumentParser

`pdb2pqr.main.`**`check_files`**`(`*args*`)`

Check for other necessary files.

Parameters **`args`** (*`argparse.Namespace`*) – command-line arguments

Raises

- **`FileNotFoundError`** – necessary files not found
- **`RuntimeError`** – input argument or file parsing problems

`pdb2pqr.main.`**`check_options`**`(`*args*`)`

Sanity check options.

Parameters **`args`** (*`argparse.Namespace`*) – command-line arguments

Raises **`RuntimeError`** – silly option combinations were encountered.

`pdb2pqr.main.`**`drop_water`**`(`*pdblist*`)`

Drop waters from a list of PDB records.

Todo: this module is already too long but this function fits better here. Other possible place would be utilities.

Parameters **`pdb_list`** (*`[str]`*) – list of PDB records as returned by io.get_molecule

Returns new list of PDB records with waters removed.

Return type [str]

`pdb2pqr.main.`**`dx_to_cube`**`()`

Convert DX file format to Cube file format.

The OpenDX file format is defined at <https://www.idvbook.com/wp-content/uploads/2010/12/opendx.pdf‘ and the Cube file format is defined at <https://docs.chemaxon.com/display/Gaussian_Cube_format.html>.

Todo: This function should be moved into the APBS code base.

`pdb2pqr.main.`**`is_repairable`**(*biomolecule*, *has_ligand*)

Determine if the biomolecule can be (or needs to be) repaired.

Parameters

- **`biomolecule`** (*`biomol.Biomolecule`*) – biomolecule object
- **`has_ligand`** (*`bool`*) – does the system contain a ligand?

Returns indication of whether biomolecule can be repaired

Return type bool

Raises **`ValueError`** – if there are insufficient heavy atoms or a significant part of the biomolecule is missing

`pdb2pqr.main.`**`main`**()

Hook for command-line usage.

`pdb2pqr.main.`**`main_driver`**(*args*)

Main driver for running program from the command line.

Validate inputs, launch PDB2PQR, handle output.

Parameters **`args`** (*`argparse.Namespace`*) – command-line arguments

`pdb2pqr.main.`**`non_trivial`**(*args*, *biomolecule*, *ligand*, *definition*, *is_cif*)

Perform a non-trivial PDB2PQR run.

Todo: These routines should be generalized to biomolecules; none of them are specific to biomolecules.

Parameters

- **`args`** (*`argparse.Namespace`*) – command-line arguments
- **`biomolecule`** (`Biomolecule`) – biomolecule
- **`ligand`** (`Mol2Molecule`) – ligand object or None
- **`definition`** (`Definition`) – topology definition
- **`is_cif`** (*`bool`*) – indicates whether file is CIF format

Raises **`ValueError`** – for missing atoms that prevent debumping

Returns dictionary with results

Return type dict

`pdb2pqr.main.`**`print_pdb`**(*args*, *pdb_lines*, *header_lines*, *missing_lines*, *is_cif*)

Print PDB-format output to specified file

Todo: Move this to another module (io)

Parameters

- **args** (*argparse.Namespace*) – command-line arguments
- **pdb_lines** (*[str]]*) – output lines (records)
- **header_lines** (*[str]*) – header lines
- **missing_lines** (*[str]*) – lines describing missing atoms (should go in header)
- **is_cif** (*bool*) – flag indicating CIF format

pdb2pqr.main.**print_pqr**(*args*, *pqr_lines*, *header_lines*, *missing_lines*, *is_cif*)

Print PQR-format output to specified file

Todo: Move this to another module (io)

Parameters

- **args** (*argparse.Namespace*) – command-line arguments
- **pqr_lines** (*[str]*) – output lines (records)
- **header_lines** (*[str]*) – header lines
- **missing_lines** (*[str]*) – lines describing missing atoms (should go in header)
- **is_cif** (*bool*) – flag indicating CIF format

pdb2pqr.main.**print_splash_screen**(*args*)

Print argument overview and citation information.

Parameters **args** (*argparse.Namespace*) – command-line arguments

pdb2pqr.main.**run_propka**(*args*, *biomolecule*)

Run a PROPKA calculation.

Parameters

- **args** (*argparse.Namespace*) – command-line arguments
- **biomolecule** (Biomolecule) – biomolecule object

Returns (DataFrame of assigned pKa values, pKa information from PROPKA)

Return type (pandas.DataFrame, str)

pdb2pqr.main.**setup_molecule**(*pdblist*, *definition*, *ligand_path*)

Set up the molecular system.

Parameters

- **pdblist** (*list*) – list of PDB records
- **definition** (Definition) – topology definition
- **ligand_path** (*str*) – path to ligand (may be None)

Returns (biomolecule object, definition object–revised if ligand was parsed, ligand object–may be None)

Return type (*Biomolecule*, *Definition*, Ligand) pdb2pqr.main.**transform_arguments**(*args*)

Transform arguments with logic not provided by argparse.

Todo: I wish this could be done with argparse.

Parameters **args** (*argparse.Namespace*) – command-line arguments

Returns modified arguments

Return type argparse.Namespace

psize

Get dimensions and other information from a PQR file.

Todo: This code could be combined with `inputgen`.

Todo: This code should be moved to the APBS code base.

Code author: Dave Sept

Code author: Nathan Baker

Code author: Todd Dolinksy

Code author: Yong Huang

class pdb2pqr.psize.**Psize**(*cfac=1.7*, *fadd=20.0*, *space=0.5*, *gmemfac=200*, *gmemceil=400*, *ofrac=0.1*, *redfac=0.25*)

Class for parsing input files and suggesting settings.

__init__(*cfac=1.7*, *fadd=20.0*, *space=0.5*, *gmemfac=200*, *gmemceil=400*, *ofrac=0.1*, *redfac=0.25*)

Initialize.

Parameters

- **cfac** (*float*) – factor by which to expand molecular dimensions to get coarse grid dimensions
- **fadd** (*float*) – amount (in Angstroms) to add to molecular dimensions to get the fine grid dimensions
- **space** (*float*) – desired fine mesh resolution (in Angstroms)
- **gmemfac** (*float*) – number of bytes per grid point required for a sequential multigrid calculation
- **gmemceil** (*float*) – maximum memory (in MB) allowed for sequential multigrid calculation. Adjust this value to force the script to perform faster calculations (which require more parallelism).
- **ofrac** (*float*) – overlap factor between mesh partitions in parallel focusing calculation
- **redfac** (*float*) – the maximum factor by which a domain dimension can be reduced during focusing

parse_input (*filename*)
Parse input structure file in PDB or PQR format.

Parameters **filename** (*str*) – string with path to PDB- or PQR-format file.

parse_lines (*lines*)
Parse the PQR/PDB lines.

Todo: This is messed up. Why are we parsing the PQR manually here when we already have other routines to do that? This function should be replaced by a call to existing routines.

Parameters **lines** (*[str]*) – PDB/PQR lines to parse

parse_string (*structure*)
Parse the input structure as a string in PDB or PQR format.

Parameters **structure** (*str*) – input structure as string in PDB or PQR format.

run_psize (*filename*)
Parse input PQR file and set parameters.

Parameters **filename** (*str*) – path of PQR file

set_all ()
Set up all of the things calculated individually above.

set_center (*maxlen*, *minlen*)
Compute molecular center.

Parameters

- **maxlen** (*[float, float, float]*) – maximum molecule lengths
- **minlen** (*[float, float, float]*) – minimum molecule lengths

Returns center of molecule

Return type [float, float, float]

set_coarse_grid_dims (*mol_length*)
Compute coarse mesh lengths.

Parameters **mol_length** (*[float, float, float]*) – input molecule lengths

Returns coarse grid dimensions

Return type [float, float, float]

set_fine_grid_dims (*mol_length*, *coarse_length*)
Compute fine mesh lengths.

Parameters

- **mol_length** (*[float, float, float]*) – input molecule lengths
- **coarse_length** (*[float, float, float]*) – coarse grid lengths

Returns fine grid lengths

Return type [float, float, float]

set_fine_grid_points (*fine_length*)
Compute mesh grid points, assuming 4 levels in multigrid hierarchy.

Todo: remove hard-coded values from this function.

Parameters **fine_length** (*[float, float, float]*) – lengths of the fine grid

Returns number of grid points in each direction

Return type [int, int, int]

set_focus (*fine_length*, *nproc*, *coarse_length*)

Calculate the number of levels of focusing required for each processor subdomain.

Parameters

- **fine_length** (*[float, float, float]*) – fine grid length
- **nproc** (*[int, int, int]*) – number of processors in each dimension
- **coarse_length** (*[float, float, float]*) – coarse grid length

set_length (*maxlen*, *minlen*)

Compute molecular dimensions, adjusting for zero-length values.

Todo: Replace hard-coded values in this function.

Parameters

- **maxlen** (*[float, float, float]*) – maximum dimensions from molecule
- **minlen** (*[float, float, float]*) – minimum dimensions from molecule

Returns molecular dimensions

Return type [float, float, float]

set_proc_grid (*ngrid*, *nsmall*)

Calculate the number of processors required in a parallel focusing calculation to span each dimension of the grid given the grid size suitable for memory constraints.

Parameters

- **ngrid** (*[int, int, int]*) – number of needed grid points
- **nsmall** (*[int, int, int]*) – number of grid points that will fit in memory

Returns number of processors needed in each direction

Return type [int, int, int]

set_smallest (*ngrid*)

Set smallest dimensions.

Compute parallel division of domain in case the memory requirements for the calculation are above the memory ceiling. Find the smallest dimension and see if the number of grid points in that dimension will fit below the memory ceiling Reduce nsmall until an nsmall^3 domain will fit into memory.

Todo: Remove hard-coded values from this function.

Parameters **ngrid** (*[int, int, int]*) – number of grid points

Returns smallest number of grid points in each direction to fit in memory

Return type [int, int, int]

`pdb2pqr.psize.`**`build_parser`**`()`

Build argument parser.

Returns argument parser

Return type argparse.ArgumentParser

`pdb2pqr.psize.`**`main`**`()`

Main driver for module.

quatfit

Quatfit routines for PDB2PQR

This module is used to find the coordinates of a new atom based on a reference set of coordinates and a definition set of coordinates.

Original Code by David J. Heisterberg, The Ohio Supercomputer Center, 1224 Kinnear Rd., Columbus, OH 43212-1163, (614)292-6036, djh@osc.edu, djh@ohstpy.bitnet, ohstpy::djh

Translated to C from fitest.f program and interfaced with Xmol program by Jan Labanowski, jkl@osc.edu, jkl@ohstpy.bitnet, ohstpy::jkl

Todo: There are many unnecessary parameters in this module due to FORTRAN/C assumptions about how the code should behave.

Code author: David Heisterberg

Code author: Jan Labanowski

Code author: Jens Erik Nielsen

Code author: Todd Dolinsky

`pdb2pqr.quatfit.`**`center`**`(`*numpoints*, *refcoords*`)`

Center a molecule using equally weighted points.

Parameters

- **`numpoints`** (*`int`*) – number of points
- **`refcoords`** (*`[[float, float, float]]`*) – list of reference coordinates, with each set a list of form [x,y,z]

Returns (center of the set of points, moved refcoords relative to refcenter)

`pdb2pqr.quatfit.`**`find_coordinates`**`(`*numpoints*, *refcoords*, *defcoords*, *defatomcoords*`)`

Driver for the quaternion file.

Provide the coordinates as inputs and obtain the coordinates for the new atom as output.

Parameters

- **`numpoints`** (*`int`*) – the number of points in each list
- **`refcoords`** (*`[[float, float, float]]`*) – the reference coordinates, a list of lists of form [x,y,z]

- **defcoords** (*[[float, float, float]]*) – the definition coordinates, a list of lists of form [x,y,z]
- **defatomcoords** (*[[float, float, float]]*) – the definition coordinates for the atom to be placed in the reference frame

Returns the coordinates of the new atom in the reference frame

Return type [[float, float, float]]

`pdb2pqr.quatfit.`**`jacobi`**(*amat*, *nrot*)

Jacobi diagonalizer with sorted output, only good for 4x4 matrices.

Parameters

- **amat** – Matrix to diagonalize
- **nrot** (*int*) – maximum number of sweeps

Returns (eigenvalues, eigenvectors)

`pdb2pqr.quatfit.`**`q2mat`**(*quat*)

Generate a left rotation matrix from a normalized quaternion

Parameters **quat** (*[[float, float, float, float]]*) – the normalized quaternion

Returns the rotation matrix

`pdb2pqr.quatfit.`**`qchichange`**(*initcoords*, *refcoords*, *angle*)

Change the chiangle of the reference coordinate.

Change the chiangle of the reference coordinate using the initcoords and the given angle.

Parameters

- **initcoords** (*[[float, float, float]]*) – coordinates based on the point and basis atoms (one-dimensional list)
- **difchi** (*float*) – the angle to use
- **refcoords** (*[[float, float, float]]*) – the atoms to analyze (list of many coordinates)

Returns the new coordinates of the atoms

Return type [[float, float, float]]

`pdb2pqr.quatfit.`**`qfit`**(*numpoints*, *refcoords*, *defcoords*)

Method for getting new atom coordinates from sets of reference and definition coordinates.

Todo: Remove hard-coded parameters of function.

Parameters

- **numpoints** (*int*) – the number of points in each list
- **refcoords** (*[[float, float, float]]*) – list of reference coordinates
- **defcoords** (*[[float, float, float]]*) – list of definition coordinates

Returns (reference center, definition center, left rotation matrix)

Return type ([[float, float, float]], [[float, float, float]], [[float, float, float]]) pdb2pqr.quatfit.**qtransform**(*numpoints*, *defcoords*, *refcenter*, *fitcenter*, *rotation*)

Transform coordinates using the reference.

Transform the set of defcoords using the reference center, the fit center, and a rotation matrix.

Parameters

- **numpoints** (*int*) – the number of points in each list
- **defcoords** (*[[float, float, float]]*) – set of coordinates to be transformed using the reference center and a rotation matrix
- **refcenter** (*[[float, float, float]]*) – the reference center
- **fitcenter** (*[float, float, float]*) – the definition center
- **rotation** (*[[float, float, float]]*) – the rotation matrix

Returns the coordinates of the new point

Return type [[float, float, float]]

pdb2pqr.quatfit.**qtrfit**(*numpoints*, *defcoords*, *refcoords*, *nrot*)

Find the best-fit quaternion.

Find the quaternion, q, [and left rotation matrix, u] that minimizes

$$|qTXq - Y|^2[|uX - Y|^2]$$

This is equivalent to maximizing

$$Re(q^TX^TqY)$$

The left rotation matrix, u, is obtained from q by

$$u = qT1q$$

Parameters

- **numpoints** (*int*) – the number of points in each list
- **defcoords** (*[[float, float, float]]*) – list of definition coordinates, with each set a list of form [x,y,z]
- **refcoords** (*[[float, float, float]]*) – list of fitted coordinates, with each set a list of form [x,y,z]
- **nrot** (*int*) – the maximum number of Jacobi sweeps

Returns (the best-fit quaternion, the best-fit left rotation matrix)

pdb2pqr.quatfit.**rotmol**(*numpoints*, *coor*, *lrot*)

Rotate a molecule

Parameters

- **numpoints** (*int*) – the number of points in the list
- **coor** (*[[float, float, float]]*) – the input coordinates
- **lrot** (*[[float, float, float]]*) – the left rotation matrix

Returns the rotated coordinates

Return type [[float, float, float]]

pdb2pqr.quatfit.**translate**(*numpoints*, *refcoords*, *center_*, *mode*)

Translate a molecule using equally weighted points.

Parameters

- **numpoints** (*int*) – number of points
- **refcoords** – list of reference coordinates, with each set a list of form [x,y,z]
- **center** (*[float, float, float]*) – center of the system
- **mode** – if 1, center will be subtracted from refcoords; if 2, center will be added to refcoords

Type list

Returns moved refcoords relative to refcenter

Return type [[float, float, float]]

run

Routines for running the code with a given set of options and PDB files.

pdb2pqr.run.**run_pdb2pka**(*ph*, *force_field*, *pdb_list*, *ligand*, *pdb2pka_params*)

Run PDB2PKA

pdb2pqr.run.**run_pdb2pqr**(*pdblist*, *my_biomolecule*, *my_definition*, *options*, *is_cif*)

Run the PDB2PQR Suite

utilities

Utilities for the PDB2PQR software suite.

Code author: Todd Dolinsky

Code author: Yong Huang

Code author: Nathan Baker

pdb2pqr.utilities.**add**(*coords1*, *coords2*)

Add one 3-dimensional point to another.

Parameters

- **coords1** (*[float, float, float]*) – coordinates of form [x,y,z]
- **coords2** (*[float, float, float]*) – coordinates of form [x,y,z]

Returns list of coordinates equal to coords2 + coords1

Return type numpy.ndarray

pdb2pqr.utilities.**analyze_connectivity**(*map_*, *key*)

Analyze the connectivity of a given map using the key value.

Parameters

- **map** (*dict*) – map to analyze
- **key** (*str*) – key value

Returns list of connected values to the key

Return type list

`pdb2pqr.utilities.`**`angle`**`(`*coords1*, *coords2*, *coords3*`)`

Get the angle between three coordinates.

Parameters

- **`coords1`** (*`[float, float, float]`*) – first coordinate set
- **`coords2`** (*`[float, float, float]`*) – second (vertex) coordinate set
- **`coords3`** (*`[float, float, float]`*) – third coordinate set

Returns angle between the atoms (in degrees)

Return type float

`pdb2pqr.utilities.`**`cross`**`(`*coords1*, *coords2*`)`

Find the cross-product of one 3-dimensional point with another.

Parameters

- **`coords1`** (*`[float, float, float]`*) – coordinates of form [x,y,z]
- **`coords2`** (*`[float, float, float]`*) – coordinates of form [x,y,z]

Returns list of coordinates equal to coords2 cross coords1

Return type numpy.ndarray

`pdb2pqr.utilities.`**`dihedral`**`(`*coords1*, *coords2*, *coords3*, *coords4*`)`

Calculate the dihedral angle from four atoms' coordinates.

Parameters

- **`coords1`** (*`[float, float, float]`*) – one of four coordinates of form [x,y,z]
- **`coords2`** (*`[float, float, float]`*) – one of four coordinates of form [x,y,z]
- **`coords3`** (*`[float, float, float]`*) – one of four coordinates of form [x,y,z]
- **`coords4`** (*`[float, float, float]`*) – one of four coordinates of form [x,y,z]

Returns the angle (in degrees)

Return type float

`pdb2pqr.utilities.`**`distance`**`(`*coords1*, *coords2*`)`

Calculate the distance between two coordinates.

Parameters

- **`coords1`** (*`[float, float, float]`*) – coordinates of form [x,y,z]
- **`coords2`** (*`[float, float, float]`*) – coordinates of form [x,y,z]

Returns distance between the two coordinates

Return type float

`pdb2pqr.utilities.`**`dot`**`(`*coords1*, *coords2*`)`

Find the dot-product of one 3-dimensional point with another.

Parameters

- **`coords1`** (*`[float, float, float]`*) – coordinates of form [x,y,z]
- **`coords2`** (*`[float, float, float]`*) – coordinates of form [x,y,z]

Returns list of coordinates equal to the inner product of coords2 with coords1

Return type numpy.ndarray

`pdb2pqr.utilities.`**`factorial`**`(`*num*`)`

Returns the factorial of the given number.

Parameters **`num`** (*`int`*) – number for which to compute factorial

Returns factorial of number

Return type int

`pdb2pqr.utilities.`**`normalize`**`(`*coords*`)`

Normalize a set of coordinates to unit vector.

Parameters **`coords`** (*`[float, float, float]`*) – coordinates of form [x,y,z]

Returns normalized coordinates

Return type numpy.ndarray

`pdb2pqr.utilities.`**`shortest_path`**`(`*graph*, *start*, *end*, *path=[]*`)`

Find the shortest path between two nodes.

Uses recursion to find the shortest path from one node to another in an unweighted graph. Adapted from http://www.python.org/doc/essays/graphs.html

Parameters

- **`graph`** (*`dict`*) – a mapping of the graph to analyze, of the form {0: [1,2], 1:[3,4], . . . } . Each key has a list of edges.
- **`start`** (*`str`*) – the ID of the key to start the analysis from
- **`end`** (*`str`*) – the ID of the key to end the analysis
- **`path`** (*`list`*) – optional argument used during the recursive step to keep the current path up to that point

Returns list of the shortest path or `None` if start and end are not connected

Return type list

`pdb2pqr.utilities.`**`sort_dict_by_value`**`(`*inputdict*`)`

Sort a dictionary by its values.

Parameters **`inputdict`** (*`dict`*) – the dictionary to sort

Returns list of keys sorted by value

Return type list

`pdb2pqr.utilities.`**`subtract`**`(`*coords1*, *coords2*`)`

Suntract one 3-dimensional point from another.

Parameters

- **`coords1`** (*`[float, float, float]`*) – coordinates of form [x,y,z]
- **`coords2`** (*`[float, float, float]`*) – coordinates of form [x,y,z]

Returns list of coordinates equal to coords2 - coords1

Return type numpy.ndarray

2.8 Release history

2.8.1 3.1.0 (2020-12-22)

Additions

- Created Sphinx documentation of usage and API at http://pdb2pqr.readthedocs.io (#88, #90).
- New command line tools added with documentation (#163).
- Added support for reading QCD-format structure files (#137).
- Added versioneer support for versioning (#104).
- Made several APBS tools available as PDB2PQR scripts: `dx2cube` (#98), `inputgen` (#105), `psize` (#106).
- Added code of conduct document (#62).

Fixes

- Fixed faulty no-op logic in debumping routines (#162)
- Fixed problem with element type in PDB output (#159)
- Updated very out-of-date change log (#153).
- Fixed atom-ordering problem in PDB output (#134).
- Fixed REDVAT PDB record parsing (#119).
- Fixed broken `--apbs-input` option (#94).
- Fixed OS-specific file handing (#78).

Changes

- PDB2PKA is still removed from the code base while refactoring for a code base that is more friendly to multiple platforms.
- Added Python 3.9 to testing (#161).
- Enabled additional PROPKA output (#143).
- Moved mmCIF support to external module `mmcif-pdbx` (#135).
- Added formal PQR parser (#97).
- Made failure due to missing backbone atoms more graceful (#95).
- Moved some logging output from stdout/stderr to files (#74).
- Increased testing (#70, #73).
- Continued de-linting and refactoring (#56, #122).

2.8.2 3.0.1 (2020-07-03)

Fixes

- Fixed packaging problem

2.8.3 3.0.0 (2020-07-03)

Additions

- Added ability to read mmCIF files.

Fixes

- Updated URL used to fetch PDB files from RCSB.
- Fixed naming error for CYS hydrogen.
- Replaced Python pickle with portable JSON.

Changes

- Upgraded to Python 3.
- Changed primary distribution mechanism into Python package (#45)
- Upgraded web interface.
- Upgraded to PROPKA 3.1 (and converted to `pip` dependency rather than submodule).
- Removed PDB2PKA support.
- Added coverage tests to testing.
- Removed support for extensions.
- Significant code refactoring.
- Changed output from `print()` to `logging`.
- Provided additional warnings when dropping HETATM entries.
- Improved build system.
- Increased list of proteins used in testing.
- Removed Opal support.
- Added GitHub actions for continuous integration testing.

2.8.4 2.1.1 (2016-03)

Additions

- Replaced the Monte Carlo method for generating titration curves with Graph Cut. See http://arxiv.org/1507.07021/

Fixes

- Added a check before calculating pKa's for large interaction energies

Known bugs

- If more than one extension is run from the command line and one of the extensions modifies the protein data structure it could affect the output of the other extension. The only included extensions that exhibit this problem are resinter and newresinter.
- Running ligands and PDB2PKA at the same time is not currently supported.
- PDB2PKA currently leaks memory slowly. Small jobs will use about twice the normally required RAM (i.e. ~14 titratable residues will use 140MB). Big jobs will use about 5 times the normally required RAM (60 titratable residues will use 480MB). We are working on this.

2.8.5 2.1.0 (2015-12)

Additions

- Added alternate method to do visualization using 3dmol.
- Replaced the Monte Carlo method for generating titration curves with Graph Cut. See http://arxiv.org/abs/1507.07021. If you prefer the Monte Carlo Method, please use http://nbcr-222.ucsd.edu/pdb2pqr_2.0.0/

Fixes

- Added compile options to allow for arbitrary flags to be added. Helps work around some platforms where scons does not detect the needed settings correctly.
- Fixed broken links on APBS submission page.
- Added some missing files to query status page results.
- Fixed some pages to use the proper CSS file.
- Better error message for `--assign-only` and HIS residues.
- Fixed PROPKA crash for unrecognized residue.
- Debumping routines are now more consistent across platforms. This fixes pdb2pka not giving the same results on different platforms.

Changes

- Added `fabric` script used to build and test releases.
- The `newtworkx` library is now required for `pdb2pka`.

Known bugs

- If more than one extension is run from the command line and one of the extensions modifies the protein data structure it could affect the output of the other extension. The only included extensions that exhibit this problem are resinter and newresinter.
- Running ligands and PDB2PKA at the same time is not currently supported.
- PDB2PKA currently leaks memory slowly. Small jobs will use about twice the normally required RAM (i.e. ~14 titratable residues will use 140MB). Big jobs will use about 5 times the normally required RAM (60 titratable residues will use 480MB). We are working on this.

2.8.6 2.0.0 (2014-12)

Additions

- Improved look of web interface.
- Option to automatically drop water from pdb file before processing.
- Integration of PDB2PKA into PDB2PQR as an alternative to PROPKA.
- Support for compiling with VS2008 in Windows.
- Option to build with debug headers.
- PDB2PKA now detects and reports non Henderson-Hasselbalch behavior.
- PDB2PKA can be instructed whether or not to start from scratch with `--pdb2pka-resume`.
- Can now specify output directory for PDB2PKA.
- Improved error regarding backbone in some cases.
- Changed time format on query status page.
- Improved error catching on web interface.

Fixes

- Fixed executable name when creating binaries for Unix based operating systems.
- Fixed potential crash when using `--clean` with extensions.
- Fixed MAXATOMS display on server home page.
- PDB2PKA now mostly respects the `--verbose` setting.
- Fixed how hydrogens are added by PDB2PKA for state changes in some cases.
- Fixed **:mode:'psize'** error check.
- Will now build properly without ligand support if `numpy` is not installed.
- Removed old automake build files from all test ported to scons.
- Fixed broken opal backend.

Changes

- Command line interface to PROPKA changed to accommodate PDB2PKA. PROPKA is now used with `--ph-calc-method=propka --with-ph` now defaults to 7.0 and is only required if a different pH value is required.
- `--ph-calc-method` to select optional method to calculate pH values used to protonate titratable residues. Possible options are "propka" and "pdb2pka".
- Dropped support for compilation with mingw. Building on Windows now requires VS 2008 installed in the default location.
- Updated included Scons to 2.3.3
- PDB2PKA can now be run directly (not integrated in PDB2PQR) with pka.py. Arguments are PDBfile and Output directory.

- No longer providing 32-bit binary build. PDB2PKA support is too memory-intensive to make this practical in many cases.

Known bugs

- If more than one extension is run from the command line and one of the extensions modifies the protein data structure it could affect the output of the other extension. The only included extensions that exhibit this problem are resinter and newresinter.
- Running ligands and PDB2PKA at the same time is not currently supported.
- PDB2PKA currently leaks memory slowly. Small jobs will use about twice the normally required RAM (i.e. ~14 titratable residues will use 140MB). Big jobs will use about 5 times the normally required RAM (60 titratable residues will use 480MB). We are working on this.

2.8.7 1.9 (2014-03)

Additions

- Added support for reference command line option for PROPKA.
- Added newresinter plugin to provide alternate methods for calculating interaction energies between residues.
- Added propka support for phosphorous sp3. Thanks to Dr. Stefan Henrich

Fixes

- Rolled back change that prevented plugins from interfering with each other. Large proteins would cause a stack overflow when trying to do a deep copy
- Fixed apbs input file to match what web interface produces.
- Fixed user specified mobile ion species not being passed to apbs input file.
- Removed ambiguous A, ADE, C, CYT, G, GUA, T, THY, U, URA as possible residue names.
- Fixed hbond extension output to include insertion code in residue name.
- Fixed debumping routines not including water in their checks. Fixes bad debump of ASN B 20 in 1gm9 when run with pH 7.0.
- Fixed debumping failing to use best angle for a specific dihedral angle when no tested angles are without conflict.
- Fixed debumping using asymmetrical cutoffs and too large cutoffs in many checks involving hydrogen.
- Fixed debumping accumulating rounding error while checking angles.
- Fixed inconsistencies in pdb parsing. Thanks to Dr. Stefan Henrich
- Fixed problems with propka handling of aromatic carbon/nitrogen. Thanks to Dr. Stefan Henrich
- Fixed case where certain apbs compile options would break web visualization.
- Fixed improper handling of paths with a '.' or filenames with more than one '.' in them.

Changes

- Updated INSTALL file to reflect no more need for Fortran.
- Removed eval from pdb parsing routines.
- Updated web links where appropriate.
- Binary builds do not require python or numpy be installed to use. Everything needed to run PDB2PQR is included. Just unpack and use.
- OSX binaries require OSX 10.6 or newer. The OSX binary is 64-bit.
- Linux binaries require CentOS 6 or newer and have been tested on Ubuntu 12.04 LTS and Linux Mint 13. If you are running 64-bit Linux use the 64-bit libraries. In some cases the needed 32-bit system libraries will not be installed on a 64-bit system.
- Windows binaries are 32 bit and were built and tested on Windows 7 64-bit but should work on Windows XP, Vista, and 8 both 32 and 64-bit systems.
- PDB2PQR can now be compiled and run on Windows using MinGW32. See http://mingw.org/ for details.
- PDB2PQR now uses Scons for compilations. With this comes improved automated testing.
- A ligand file with duplicate atoms will cause pdb2pqr to stop instead of issue a warning. Trust us, this is a feature, not a bug!
- Improved error reporting.
- Mol2 file handling is now case insensitive with atom names.
- PROPKA with a pH of 7 is now specified by default on the web service.
- Compilation is now done with scons.
- Verbose output now includes information on all patches applied during a run.
- Added stderr and stdout to web error page.
- Added warning to water optimization when other water is ignored.
- Command line used to generate a pqr is now duplicated in the comments of the output.
- Added support for NUMMDL in parser.
- Added complete commandline feature test. Use complete-test target.
- Added a PyInstaller spec file. Standalone pdb2pqr builds are now possible.
- Removed `numpy` from contrib. The user is expected to have `numpy` installed and available to python at configuration.
- Support for `numeric` dropped.

Known bugs

- If more than one extension is run from the command line and one of the extensions modifies the protein data structure it could affect the output of the other extension. The only included extensions that exhibit this problem are resinter and newresinter.

2.8.8 1.8 (2012-01)

Additions

- Added residue interaction energy extension
- Added Opal configuration file.

Fixes

- Cleaned up white space in several files and some pydev warnings
- Creating print output no longer clears the chain id data from atoms in the data. (Affected resinter plugin)
- Removed possibility of one plug-in affecting the output of another
- Fixed `--protonation=new` option for `propka30`
- Improved time reporting for apbs jobs
- Fixed opal runtime reporting
- Fixed misspelled command line options that prevented the use of PEOEPB and TYL06
- Fixed error handling when certain data files are missing
- Fixed **`LDFLAGS`** environment variable not being used along with python specific linker flags to link `Algorithms.o` and `_pMC_mult.so`
- Fixed possible Attribute error when applying naming scheme.

Changes

- Updated PROPKA to version 3.0
- Added protein summary extension
- Combined `hbond` and `hbondwhatif` into one extension (`hbond`) with new command line parameters
- Combined `rama`, `phi`, `psi` into one extension (`rama`) with new command line parameters.
- Extensions may now add their own command line arguments. Extensions with their own command line arguments will be grouped separately.
- Improved interface for extensions

2.8.9 1.7.1a (2011-09-13)

Additions

- Added force field example.

Fixes

- Fixed ligand command line option.
- Fixed capitalization of force field in PQR header.
- Fixed error handling for opal errors.

- Fixed web logging error when using ligand files, user force fields, and name files.
- Fixed extension template in documentation.
- Fixed 1a1p example README to reflect command line changes.

2.8.10 1.7.1 (2011-08)

Additions

- Switched Opal service urls from sccne.wustl.edu to NBCR.
- Added more JMol controls for visualization, JMol code and applets provided by Bob Hanson.
- Changed default forcefield to PARSE in web interface.

Fixes

- Fixed crash when opal returns an error.
- Fixed specific combinations of command-line arguments causing `pdb2pqr.py` to crash.
- Fixed opal job failing when filenames have spaces or dashs.
- Fixed gap in backbone causing irrationally placed hydrogens.
- Fixed crash when too many fixes are needed when setting termini.
- Corrected web and command line error handling in many cases.
- Fixed `--username` command line option.
- Fixed ambiguous user created forcefield and name handling. Now `--username` is required if `--userff` is used.
- Fixed `querystatus.py` not redirecting to generated error page.

2.8.11 1.7 (2010-10)

Changes

- For PDB2PQR web interface users: the JMol web interface for APBS calculation visualization has been substantially improved, thanks to help from Bob Hanson. Those performing APBS calculations via the PDB2PQR web interface now have a much wider range of options for visualizing the output online – as well as downloading for offline analysis.
- For PDB2PQR command-line and custom web interface users: the Opal service URLs have changed to new NBCR addresses. Old services hosted at .wustl.edu addresses have been decommissioned. Please upgrade ASAP to use the new web service. Thank you as always to the staff at NBCR for their continuing support of APBS/PDB2PQR web servers and services.

2.8.12 1.6 (2010-04)

Additions

- Added Swanson force field based on Swanson et al paper (http://dx.doi.org/10.1021/ct600216k).

- Modified `printAtoms()` method. Now "TER" is printed at the end of every chain.
- Added Google Analytics code to get the statistics on the production server.
- Modified APBS calculation page layout to hide parameters by default and display PDB ID
- Added `make test-webserver`, which tests a long list of PDBs (246 PDBs) on the production PDB2PQR web server.
- Removed `nlev` from `inputgen.py` and `inputgen_pKa.py` as nlev keyword is now deprecated in APBS.
- Added PARSE parameters for RNA, data from: Tang C. L., Alexov E, Pyle A. M., Honig B. Calculation of pKas in RNA: On the Structural Origins and Functional Roles of Protonated Nucleotides. Journal of Molecular Biology 366 (5) 1475-1496, 2007.

Fixes

- Fixed a minor bug: when starting `pka.py` from pdb2pka directory using command like `python pka.py [options] inputfile`, we need to make sure scriptpath does not end with "/".
- Fixed a bug which caused "coercing to Unicode: need string or buffer, instance found" when submitting PDB2PQR jobs with user-defined force fields on Opal based web server.
- Fixed a bug in `main_cgi.py`, now Opal-based PDB2PQR jobs should also be logged in `usage.txt` file.
- Updated `src/utilities.py` with a bug fix provided by Greg Cipriano, which prevents infinite loops in analyzing connected atoms in certain cases.
- Fixed a bug related to neutraln and/or neutralc selections on the web server.
- Fixed a special case with `--ffout` and 1AIK, where the N-terminus is acetylated.
- Fixed a bug in `psize.py` per Michael Lerner's suggestion. The old version of `psize.py` gives wrong cglen and fglen results in special cases (e.g., all y coordinates are negative values).
- Fixed a bug in `main_cgi.py`, eliminated input/output file name confusions whether a PDB ID or a pdb file is provided on the web server.
- Fixed a bug which causes run time error on the web server when user-defined force field and names files are provided.
- Fixed a bug in `apbs_cgi.py`: pdb file names submitted by users are not always 4 characters long.

2.8.13 1.5 (2009-10)

Additions

- APBS calculations can be executed through the PDB2PQR web interface in the production version of the server
- APBS-calculated potentials can be visualized via the PDB2PQR web interface thanks to Jmol
- Disabled Typemap output by default, added –typemap flag to create typemap output if needed.
- Enabled "Create APBS Input File" by default on the web server, so that APBS calculation and visualization are more obvious to the users.
- Added warnings to stderr and the REMARK field in the output PQR file regarding multiple occupancy entries in PDB file.
- Added more informative messages in REMARK field, explaining why PDB2PQR was unable to assign charges to certain atoms.

- Added `make test-long`, which runs PDB2PQR on a long list (246) of PDBs by default, it is also possible to let it run on specified number of PDBs, e.g., `export TESTNUM=50; make test-long`
- Merged PDB2PKA code, PDB2PKA is functional now.
- Added two new options: `--neutraln` and `--neutralc`, so that users can manually make the N-termini or C-termini of their proteins neutral.
- Added a `local-test`, which addresses the issue of Debian-like Linux distros not allowing fetching PDBs from the web.
- Added deprotonated Arginine form for post-PROPKA routines. This only works for PARSE forcefield as other forcefields lack deprotonated ARG parameters.

Fixes

- Verbosity outputs should be stdouts, not stderrs in web server interface. Corrected this in `src/routines.py`.
- Fixed a bug in `psize.py`: for a pqr file with no ATOM entries but only HETATM entries in it, `inputgen.py` should still create an APBS input file with reasonable grid lengths.
- Added special handling for special mol2 formats (unwanted white spaces or blank lines in ATOM or BOND records).
- Added template file to doc directory, which fixed a broken link in programmer guide.

Changes

- Updated structures.py, now PDB2PQR keeps the insertion codes from PDB files.
- Updated NBCR opal service urls from http://ws.nbcr.net/opal/… to http://ws.nbcr.net/opal2/…
- Compressed APBS OpenDX output files in zip format, so that users can download zip files from the web server.
- Removed “EXPERIMENTAL” from APBS web solver interface and Jmol visualization interface.
- Updated all APBS related urls from http://apbs.sourceforge.net/… to http:/apbs.wustl.edu/…
- Updated inputgen.py with –potdx and –istrng options added, original modification code provided by Miguel Ortiz-Lombardía.
- Changed default Opal service from http://ws.nbcr.net/opal2/services/pdb2pqr_1.4.0 to http://sccne.wustl.edu:8082/opal2/services/pdb2pqr-1.5

2.8.14 1.4.0 (2009-03)

Additions

- Added a whitespace option by by putting whitespaces between atom name and residue name, between x and y, and between y and z.
- Added radius for Chlorine in ligff.py.
- Added PEOEPB forcefield, data provided by Paul Czodrowski.
- Updated inputgen.py to write out the electrostatic potential for APBS input file.

Fixes

- Fixed a legacy bug with the web server (web server doesn't like ligand files generated on Windows or old Mac OS platforms).
- Fixed a bug in `configure.ac`, so that PDB2PQR no longer checks for `Numpy.pth` at configure stage.
- Updated `pdb2pka/substruct/Makefile.am`.
- Fixed `isBackbone()` bug in `definitions.py`.
- Fixed a bug for `Carboxylic` residues in `hydrogens.py`.
- Fixed a bug in `routines.py`, which caused hydrogens added in LEU and ILE in eclipsed conformation rather than staggered.
- Fixed a bug in `configure.ac`, now it is OK to configure with double slashes in the prefix path, e.g., `--prefix=/foo/bar//another/path`
- Fixed a bug in nucleic acid naming scheme.
- Fixed a bug involving MET, GLY as NTERM, CTERM with `--ffout` option.
- Fixed a bug for PRO as C-terminus with PARSE forcefield.
- Fixed a bug for ND1 in HIS as hacceptor.
- Fixed the `--clean` option bug.
- Fixed a bug in CHARMM naming scheme.
- Fixed a bug in `test.cpp` of the simple test (which is related to recent modifications of 1AFS in Protein Data Bank).

Changes

- Updated `html/master-index.html`, deleted `html/index.php`.
- Updated pydoc by running `genpydoc.sh`.
- Updated CHARMM.DAT with two sets of phosphoserine parameters.
- Allowed amino acid chains with only one residue, using `--assign-only` option.
- Updated `server.py.in` so that the ligand option is also recorded in `usage.txt`.
- Updated HE21, HE22 coordinates in GLN according to the results from AMBER Leap program.
- Updated `Makefile.am` with Manuel Prinz's patch (removed distclean2 and appended its contents to distclean-local).
- Updated `configure.ac`, `pdb2pqr-opal.py`; added `AppService_client.py` and `AppService_types.py` with Samir Unni's changes, which fixed earlier problems in invoking Opal services.
- Applied two patches from Manuel Prinz to `pdb2pka/pMC_mult.h` and `pdb2pka/ligand_topology.py`.
- Updated `PARSE.DAT:file:` with the source of parameters.
- Created a `contrib` folder with `numpy-1.1.0` package. PDB2PQR will install numpy by default unless any of the following conditions is met:
  - Working version of NumPy dectected by autoconf.
  - User requests no installation with `--disable-pdb2pka` option.

  - User specifies external NumPy installation.

- Merged Samir Unni's branch. Now PDB2PQR Opal and APBS Opal services are available (through `--with-opal` and/or `--with-apbs`, `--with-apbs-opal` options at configure stage).
- Added error handling for residue name longer than 4 characters.
- Updated `hbond.py` with Mike Bradley's definitions for ANGLE_CUTOFF and DIST_CUTOFF by default.
- Removed PyXML-0.8.4, which is not required for ZSI installation.
- Updated propka error message for make adv-test – propka requires a version of Fortran compiler.
- Updated `na.py` and `PATCHES.xml` so that PDB2PQR handles three lettered RNA residue names (ADE, CYT, GUA, THY, and URA) as well.
- Updated NA.xml with HO2' added as an alternative name for H2", and H5" added as an alternative name for H5".
- Updated version numbers in html/ and doc/pydoc/ .
- Updated web server. When selecting user-defined forcefield file from the web server, users should also provide `.names` file.
- Removed http://enzyme.ucd.ie/Services/pdb2pqr/ from web server list.
- Eliminated the need for protein when processing other types (ligands, nucleic acids).
- Updated `psize.py` with Robert Konecny's patch to fix inconsistent assignment of fine grid numbers in some (very) rare cases.
- Made whitespace option available for both command line and web server versions.
- Updated `inputgen_pKa.py` with the latest version.

2.8.15 1.3.0 (2008-01)

Additions

- Added `make test` and `make adv-test`
- Added integration with Opal for launching jobs as well as querying status

Fixes

- Fixed the line feed bug. Now PDB2PQR handles different input files (`.pdb` and file:*.mol2*) created or saved on different platforms.
- Fixed `hbondwhatif` warning at start up.
- Fixed problems with `make dist`
- The default value of 7.00 for the pH on the server form is removed due to a problem with browser refershing.

Changes

- The user may use NUMPY to specify the location of NUMPY.
- Both PDB2PKA and PROPKA are enabled by default. PDB2PKA is enabled by default since ligand parameterization would fail without this option.

- For a regular user, `make install` tells the user the exact command the system administrator will use to make the URL viewable.
- Updated warning messages for lines beginning with SITE, TURN, SSBOND and LINK.
- Switched license from GPL to BSD.
- Made a new tar ball `pdb2pqr-1.3.0-1.tar.gz` for Windows users who cannot create file:*pdb2pqr.py* through configure process.
- file:*configure* now automatically detects SRCPATH, WEBSITE, and the location of file:*pdb2pqr.cgi*. In version 1.2.1, LOCALPATH(SRCPATH) and WEBSITE were defined in file:*src/server.py* and the location of file:*pdb2pqr.cgi* was specified in file:*html/server.html* (file:*index.html*). Configure now uses variable substitution with new files file:*src/server.py.in* and file:*html/server.html.in* to create file:*src/server.py* and file:*html/server.html* (file:*index.html*).
- **`SRCPATH`** is automatically set to the current working directory. **`WEBSITE`** is automatically set to http://fully_qualified_domain_name/pdb2pqr. Path to CGI is automcailly set to http://fully_qualified_domain_name/pdb2pqr/pdb2pqr.cgi.
- In version 1.2.1, there were 3 variables that needed to be changed to set up a server at a location different from agave.wustl.edu. **`LOCALPATH`**, **`WEBSITE`**, and the location of the CGI file. In this version, **`LOCALPATH`** has been used to **`SRCPATH`** to avoid confusion, since **`LOCALPATH`** could be interpreted as the local path for source files or the localpath for the server.
- Since configure now automatically sets the locations of files/directories based on the machine and configure options, the default agave.wustl.edu locations are not used anymore.
- A copy of `pdb2pqr.css` is included.
- `configure` prints out information about parameters such as python flags, srcpath, localpath, website, etc.
- `configure` now automatically creates tmp/ with r + w + x permissions.
- `configure` now automatically copies `pdb2pqr.py` to `pdb2pqr.cgi`.
- `configure` now automatically copies `html/server.html` to `index.html` after variable substitution. In `src/server.py.in` (`src/server.py`), **`WEBNAME`** is changed to `index.html`.
- `$`*`HOME`*`/pdb2pqr` is the default prefix for a regular user
- `/var/www/html` is the default prefix for root
- http://FQDN/pdb2pqr as default website.
- `make install` runs `make` first, and the copies the approprite files to `--prefix`.
- If root did not specify `--prefix` and `/var/www/html/pdb2pqr` already exists, then a warning is issued, and the user may choose to quit or overwrite that directory.
- Similary, if a regular user did not specify `--prefix` and `$`*`HOME`*`/pdb2pqr` already exists, then a warning is issued, and the user may choose to quit or overwrite that directory.
- If root does not specify `--prefix` to be a directory to be inside `/var/www/html` (for example, `--prefix=/share/apps/pdb2pqr`), then a symbolic link will be made to `/var/www/html/pdb2pqr` during `make install`.
- `configure` option `--with-url` can be specified either as something like http://sandstone.ucsd.edu/pdb2pqr-test or sandstone.ucsd.edu/pdb2pqr-test. It also doesn't matter if there's a '/' at the end.
- If user is root, and the last part of URL and prefix are different, for example, `--with-url=athena.nbcr.net/test0 --prefix=/var/www/html/pdb2pqr-test`, then a warning will be issued saying the server will be viewable from the URL specified, but not the URL based on pdb2pqr-test. In other words,

the server will be viewable from athena.abcr.net/test0, but not athena.nbcr.net/pdb2pqr-test. During `make install`, a symbolic link is created to enable users to view the server from `--with-url`.

- When making a symbolic link for root, if then link destination already exists as a directory or a symoblic link, then the user may choose to continue with creating the link and overwrite the original directory or quit.
- If the user changes **`py_path`** when running configure for PDB2PQR, then the change also applies to PROPKA.

Known issues

- The install directory name cannot contain dots.
- For python 2.2, if PDB2PQR cannot find module `sets, then :mod:`sets` needs to be copied from `.../python2.2/site-packages/MYSQLdb/sets.py` to :file:``.../lib/python2.2`

2.8.16 1.2.1 (2007-04)

Additions

- Added ligand examples to examples/ directory
- Added native support for the TYL06 forcefield. For more information on this forcefield please see Tan C, Yang L, Luo R. How well does Poisson-Boltzmann implicit solvent agree with explicit solvent? A quantitative analysis. Journal of Physical Chemistry B. 110 (37), 18680-7, 2006.
- Added a new HTML output page which relays the different atom types between the AMBER and CHARMM forcefields for a generated PQR file (thanks to the anonymous reviewers of the latest PDB2PQR paper).

Fixes

- Fixed bug where a segmentation fault would occur in PropKa if the N atom was not the first atom listed in the residue
- Fixed error message that occurred when a blank line was found in a parameter file.
- Better error handling in MOL2 file parsing.
- Fixed bug where ligands were not supported on PDB files with multiple MODEL fields.

Changes

- Updated documentation to include instructions for pdb2pka support, references, more pydoc documents.

2.8.17 1.2.0 (2007-01)

Additions

- Added new support for passing in a single ligand residue in MOL2 format via the `--ligand` command. Also available from the web server (with link to PRODRG for unsupported ligands).
- Numerous additions to examples directory (see `examples/index.html`) and update to User Guide.

Fixes

- Fixed charge assignment error when dealing with LYN in AMBER.
- Fixed crash when a chain has a single amino acid residue. The code now reports the offending chain and residue before exiting.
- Fixed hydrogen optimization bug where waters with no nearby atoms at certain orientations caused missing hydrogens.

Changes

- Added autoconf support for `pdb2pka` directory.

2.8.18 1.1.2 (2006-06)

Fixes

- Fixed a bug in the hydrogen bonding routines where PDB2PQR attempted to delete an atom that had already been deleted. (thanks to Rachel Burdge)
- Fixed a bug in chain detection routines where PDB2PQR was unable to detect multiple chains inside a single unnamed chain (thanks to Rachel Burdge)
- Fixed a second bug in chain detection routines where HETATM residues with names ending in "3" were improperly chosen for termini (thanks to Reut Abramovich)
- Fixed a bug where chains were improperly detected when only containing one HETATM residue (thanks to Reut Abramovich)

2.8.19 1.1.1 (2006-05)

Fixes

- Fixed a bug which prevented PDB2PQR from recognizing atoms from nucleic acids with "*" in their atom names. (thanks to Jaichen Wang)
- Fixed a bug in the hydrogen bonding routines where a misnamed object led to a crash for very specific cases. (thanks to Josh Swamidass)

2.8.20 1.1.0 (2006-04)

Additions

- Added an `extensions` directory for small scripts. Scripts in this directory will be automatically loaded into PDB2PQR has command line options for post-processing, and can be easily customized.
- Pydoc documentation is now included in `html/pydoc`.
- A programmer's guide has been included to explain programming decisions and ease future development.
- A `--ffout` flag has been added to allow users to output a PQR file in the naming scheme of the desired forcefield.

Fixes

- Updated `psize.py` to use centers and radii when calculating grid sizes (thanks to John Mongan)
- Fixed bug where PDB2PQR could not read PropKa results from chains with more than 1000 residues (thanks to Michael Widmann)

Changes

- Structural data files have been moved to XML format. This should make it easier for users and developers to contribute to the project.
- Code has been greatly cleaned so as to minimize values hard-coded into functions and to allow greater customizability via external XML files. This includes a more object-oriented hierarchy of structures.
- Improved detection of the termini of chains.
- Assign-only now does just that - only assigns parameters to atoms without additions, debumping, or optimizations.
- Added a `--clean` command line option which does no additions, optimizations, or forcefield assignment, but simply aligns the PDB columns on output. Useful for using post-processing scripts like those in the extensions directory without modifying the original input file.
- The `--userff` flag has been replaced by opening up the `--ff` option to user-defined files.
- User guide FAQ updated.
- The efficiency of the hydrogen bonding detection script (`--hbond`) has been greatly improved.
- Increased the number of options available to users via the PDB2PQR web server.

2.8.21 1.0.2 (2005-12)

Additions

- Added ability for users to add their own forcefield files. This should be particularly useful for HETATMs.
- Added **`sdens`** keyword to `inputgen.py` to make PDB2PQR compatibile with APBS 0.4.0.
- Added a new examples directory with a basic runthrough on how to use the various features in PDB2PQR.

Fixes

- Fixed a bug that was unable to handle N-Terminal PRO residues with hydrogens already present.
- Fixed two instances in the PropKa routines where warnings were improperly handled due to a misspelling.
- Fixed instance where chain IDs were unable to be assigned to proteins with more than 26 chains.

2.8.22 1.0.1 (2005-10)

Fixes

- Fixed a bug during hydrogen optimization that left out H2 from water if the oxygen in question had already made 3 hydrogen bonds.

Changes

- Added citation information to PQR output.

2.8.23 1.0.0 (2005-08)

This is the initial version of the PDB2PQR conversion utility. There are several changes to the various “non-official” versions previously available:

- SourceForge has been chosen as a centralized location for all things related to PDB2PQR, including downloads, mailing lists, and bug reports.
- Several additions to the code have been made, including pKa support via PropKa, a new hydrogen optimization algorithm which should increase both accuracy and speed, and general bug fixes.

2.9 Indices and tables

- genindex
- modindex
- search

Python Module Index

Index