RESPPOL Documentation

resppol

Oct 28, 2019

CONTENTS

CHAPTER

ONE

API DOCUMENTATION

1.1 The rpol module

1.2 Primary objects

This module should convert a mol2 file int a Molecule object and expose all necessary information .. todo:

```
* Load in the molecule
* Determine the equivalent atoms in the molecule
* Determine the bonds based on a smirnoff input file
```

class resppol.resppol.**Atom**(*index*, *atom_index*, *parameter_id*, *atomic_number=0*)

class resppol.resppol.**BCCPolESP**(**args, conformer=None, e_field=<Quantity([0. 0. 0.], 'elementary_charge / bohr ** 2')>*)

class resppol.resppol.**BCCUnpolESP**(**args*, *conformer=None*)

class resppol.resppol.**Conformer**(*conf*, *molecule=None*)

add_baseESP(**args*)
Adds the unpolarized molecule to this conformation.

Parameters **args** – ESPF file

1.) GESP file form g09 2.) grid.dat file and esp.dat file generated via respyte and psi4

Returns

add_polESP(**args, e_field=<Quantity([0. 0. 0.], 'bohr')>*)
Adds the unpolarized molecule to this conformation.

Parameters **args** – ESPF file

1.) GESP file form g09 2.) grid.dat file and esp.dat file generated via respyte and psi4 :param:e_field: Pint formatted electrif field e.g e_field=Q_([0.0, 0.0, 0.0], 'bohr'))

Returns

build_matrix_A()
Fast method for only optimizing charges.

Returns

build_matrix_D()
Method for building the polarization matrix D. Only used for fitting polarizations withotut charges or BCCs.

Returns

build_matrix_X()
Creates Matrix X for the RESP-POl method.

RESP and Polarization with the large matrix. Probably worth changing it to the new model.

Again the math is shown in the manuscript.

build_matrix_X_BCC()
Creates the Matrix A for the BCC-POL approach:

Parameters **mol2** – molecule class object

Returns

The math and the optimization method is explained in the current paper draft.

build_vector_B()
Creates the Vector B for the charge fitting. :return:

build_vector_C()
Vector C corresponds to matrix D and is only for pur polarization fitting.

Returns

build_vector_Y()
Creates the Vector Y for the RESP-Pol Method. :return:

build_vector_Y_BCC()
Creates vector Y for the BCC Pol method

Parameters **mol2** – molecule object

Returns

delete_distances()
Deletes the all calculated distances to free memory.

optimize_charges_alpha()
Builds the necessary matrix and vector and performs a charge and polarizabilities optimization for this 1 conformation. :return:

optimize_charges_alpha_bcc()
Builds the necessary matrix and vector and performs a charge and polarizabilities optimization for this 1 conformation. :return:

write_res_esp(*q_alpha=None*)
NOT FINISHED YET!!!

Writes the residual ESP to a file.

Parameters **qd** – list of float Point charges and polarizabilities

Returns

class resppol.resppol.**ESPGRID**

calc_esp_q_alpha(*q_alpha*)
Calculates the ESP for a given set of point charges and polarizabilities.

Parameters **qd** – list of float Point charges and polarizabilities

Returns

Calculates the ESP on every point of the initially read GESP file

calc_sse(*q_alpha*)
Calculate the Squared Sum of Errors between the stored ESP and a ESP created from qd. :param qd: list of float

Point charges and polarizabilities

Returns

get_electric_field()
Calculates the electric field at every atomic positions. :return:

sub_esp_q_alpha(*q_alpha*)
Subtracts the ESP create by a set of point charges and polarizabilities.

Parameters **qd** – list of float Point charges and polarizabilities

Returns

write_res_esp(*q_alpha*, *atoms=[]*)
NOT FINISHED YET!!!

Writes the residual ESP to a file.

Parameters **qd** – list of float Point charges and polarizabilities

Returns

class resppol.resppol.**Molecule**(*datei*, *position=0*, *trainingset=None*)
This class loads in a mol2 file and expose all relevant information. It combines the functionality of an openeye molecule with the functionality of the OFF molecule. The OE part is only needed to determine the chemical equivalent atoms. When this feature is implemented in OFF toolkit the OE molecule is not necessary anymore

Parameters **datei** – mol2 file

Returns

add_atom(*index*, *atom_index*, *parameter_id*, *atomic_number=0*)
Adds a atom object to the molecule

Parameters

- **index** –
- **atom_index** –
- **parameter_id** –

Returns

add_bond(*index*, *atom_indices*, *parameter_id*)
Adds a bond object ot the molecule

Parameters

- **index** –
- **atom_indices** –
- **parameter_id** –

Returns

add_conformer_from_mol2(*mol2file*)
Adds a conformer from a mol2 file Automatically checks if the conformer corresponds to the molecule

Parameters **mol2file** –

Returns

property chemical_eq_atoms
Something similar in compare_charges, which can be copied and slightly modified :return:

Type # Note

create_BCCmatrix_T()
Creates the bond matrix T for a molecule.

bondtyps: list of [int,int] Bonds in the set between different BCC groups

bondmatrix T: 2 dim array 1 dim atom 2 dim BCC

See also AM1-BCC paper: https://onlinelibrary.wiley.com/doi/epdf/10.1002/%28SICI%291096-987X%2820000130%2921%3A2%3C132%3A%3AAID-JCC5%3E3.0.CO%3B2-P

create_POLmatrix_R()
Create a polarization matrix. Defines which atom has which polarizability parameter.

groups: dictionary atomtyp -> int Stores the polarization group of each atom

Defines which atom belongs to the which pol group. Takes the list of atomtypes and assign the corresponding group value. Here I am using a slightly different approach in contrast to the atomtyps. as the number of different pol types is maximal the number of atom types. No pre-screening for the involved atom-types is needed

scaling(*scaleparameters=None*, *scf_scaleparameters=None*)
Takes the bond information from a molecule instances and converts it to an scaling matrix.

bonds: list of [int,int] list of atoms connected with each other

scaleparameters: [float,float,float] 1-2 scaling, 1-3 scaling, 1-4 scaling parameter

scale: matrix scaling matrix how atoms interact with each other

update_bcc(*bccs*, *alphas*)
Converts the optimized bccs and alphas to a qd object.

Parameters **mol2** – molecule class object

Returns

class resppol.resppol.**TrainingSet**(*mode='q'*, *scaleparameters=None*, *scf_scaleparameters=None*, *SCF=False*, *thole=False*, *FF='resppol/data/test_data/BCCPOL.offxml'*)
The training set class is the top level class of the resspol program.

It consist of multiple molecule instances and combines the optimization matrices and vectors across multiple molecules.

add_molecule(*datei*)
Adds a molecule. :param datei: Mol2 file of a molecule :return:

build_matrix_A()
Builds the matrix A of the underlying molecules and combines them.

This function is only used for charge optimizatoin RESP

build_matrix_X_BCC()
Builds the matrix X of the underlying molecules and combines them.

This function is only used for charge optimizatoin RESP

build_vector_Y_BCC()
Builds the matrix X of the underlying molecules and combines them.

This function is only used for charge optimizatoin RESP

load_from_file(*txtfile*)
Allows to build a TrainingSet instance from an text file. File format as followed:

Returns

resppol.resppol.**find_eq_atoms**(*mol1*)
Finds all equivalent atoms in a molecule. :return Array of pairs of equivalent atoms.

:parameter mol1: Openeye molecule object

TODO Include rdkit support for this function

1.3 The TrainingSet class

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

T

U

W