PyConSolv 1.0.0

User Manual

Contents

1. Introduction

PyConSolv is a python-based software package meant to simplify the setup for molecular dynamics (MD) simulations for transition metal containing catalysts. It interfaces and relies on freely available software packages to execute a state-of-the-art parametrization process.

2. Installation

2.1. Software Requirements

To function, PyConSolv requires the following software packages to be installed on your system:

- ORCA 5.0.x (Tested with ORCA 5.0.4)
- AmberTools (Tested with version 20-21)
- Multiwfn (Tested with version 3.8)

2.2. Python Requirements

To install PyConSolv on your system, Python 3.10 is needed, as well as the RDKit package. RDKit will be installed automatically by PyConSolv.

- Python >= 3.10
- RDKit >= v2022.09.5

2.3. Setting up the python environment

It is recommended to create a virtual environment for PyConSolv, to avoid compatibility problems. Installation is performed as shown in code snipped 1, below:

```
conda create -n pyconsolv python=3.10
conda activate pyconsolv
conda install -c conda-forge rdkit
to be added – how to install pyconsolv package
```

Code snippet 1 Conda environment creation.

3. Usage

To begin the parametrization, an XYZ (XMol format) structure of your complex should be prepared and located in an empty folder. In case a solvent which has not been parametrized is required, an empty folder should be created and an XYZ structure of your desired solvent should be placed inside. The recommended folder structure looks as shown in figure 1, below.

```
Pyconsolv_structure/
├── Structure/
│   └── input.xyz
└── Solvent/
    └── solvent_input.xyz
```

Figure 1 initial folder structure example

Once the desired folder structure is complete, start python in the PyConSolv environment and import the ConfGen package. Create a new ConfGen.PyConSolv object pointing to the input structure (input.xyz). To begin the parametrization process, execute the run function of the PyConSolv object.

A number of parameters can be passed to the run function. For details of the available keywords, please see the kewords section. While default values are provided, it is recommended to change the QM optimization method, as well as the basis set and dispersion corrections to match your needs. Likewise, it is very important to assign a total charge for your complex.

A critical aspect is the solvent. One of the pre-parametrized solvents can be used or, as mentioned above, any user-provided solvent. A simple example is provided in code snipped 2.

```
from PyConSolv import ConfGen

conf=ConfGen.PyConSolv('/abs/path/to/Pyconsolv_param/Structure/input.xyz')

*output from PyConSolv regarding version*

conf.run(method = 'BP86', basis = 'def2-SVP', dsp = 'D4' cpu=8, charge = 0, solvent = 'Acetonitrile')
```

Code snippet 2 Simple example of parametrization for a neutral compound, while changing settings for the QM method and requesting acetonitrile as a solvent. In this case, the QM method will be DFT with the BP86 functional, def2-SVP basis set, D4 dispersion corrections, using implicit solvation with acetonitrile. The calculation will be run on 8 CPU cores.

The output from PyConSolv will guide the user through the parametrization progress. There are a few steps which must take place for the parametrization, which are explained under methodology.

While the parametrization is running, a more complex folder structure is created, with each critical step being performed in a separate folder, to allow the user to investigate the output of each step in detail as well as use the input/output files for other tasks, unrelated to PyConSolv. The final folder structure looks as shown in figure 2. For a detailed explanation of the content of the files, please consult the methodology section.

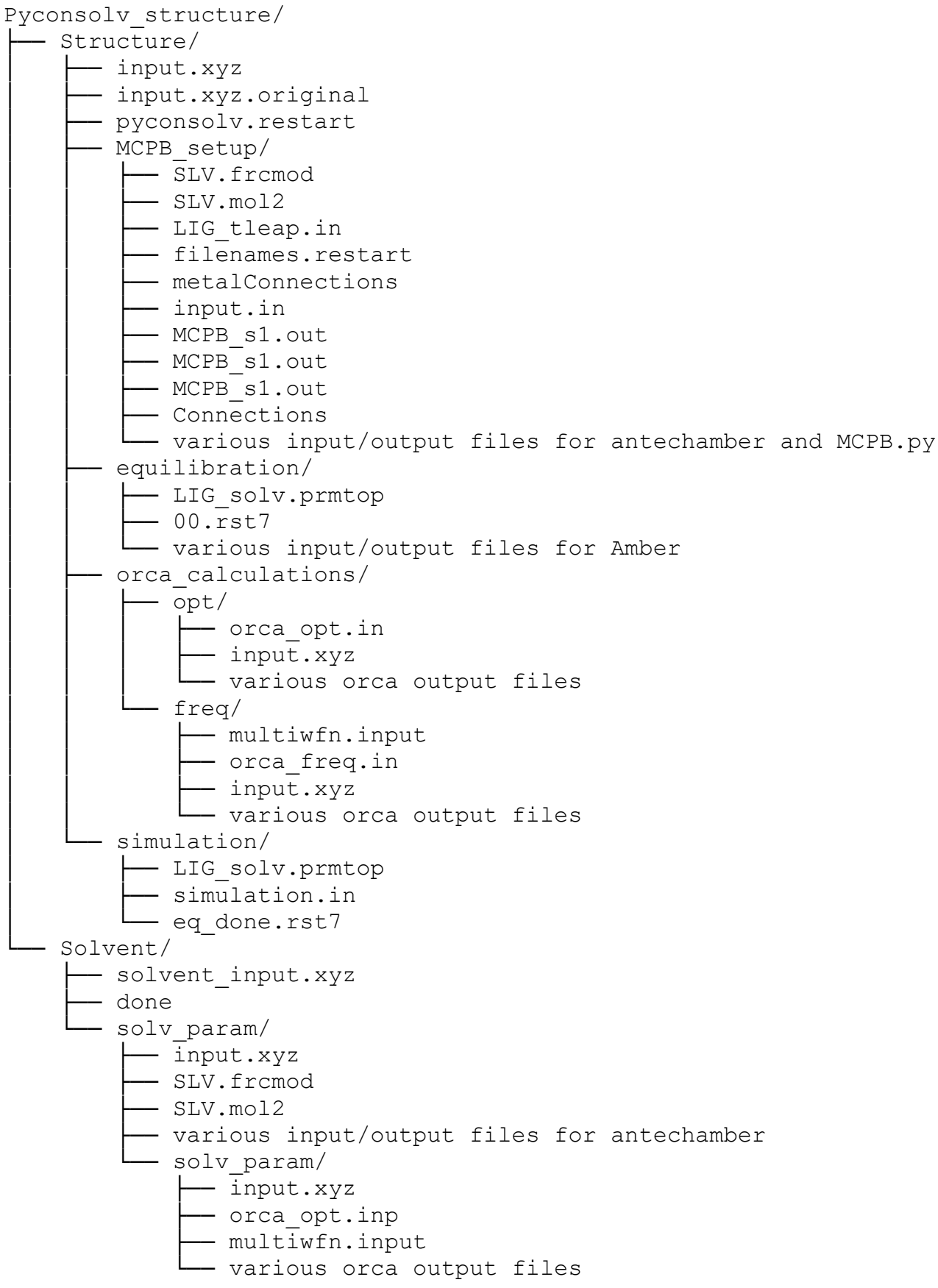

Figure 2 Folder structure and important files after parametrization is complete

4. Methodology

4.1. Program structure and files

Folder structure setup

Geometry optimization and frequency calculation

Fragment splitting

Fragment charge assignment

Generation of initial forcefield parameter files

RESP charge calculation

MCPB.py execution

Solvent box construction

Equilibration setup

4.2. Charge Fitting

4.3. MCPB.py

4.4. Amber20

4.5. Solvents

5. Keywords

5.1. Keywords for the ConfGen.PyConSolv.run() method

- **charge**: charge of the complete system; type int; default 0
- **method**: ORCA 5 method line; type string; default 'PBE0'
- **basis**: Basis set for ORCA calculations; type string; default 'def2-SVP'
- **dsp**: Dispersion corrections; type string; default 'D4'
- **cpu**: number of CPU cores to be used; type int; default 12
- **solvent**: solvent to be used for MD simulation; type string; default 'Water'.

The method, basis and dsp keywords conform to the ORCA 5 naming conventions. In theory any input that would be suitable for an orca optimization can be entered here.

Solvents can be either one of the pre-parametrized ones: Water, Acetonitrile, Acetone, Benzene, Cyclohexane, Chloroform, CCl4, CH2Cl2, DMF, DMSO, Ethanol, Hexane, Methanol, Ammonia, Octanol, THF, Toluene or can take the value 'custom'. When the value 'custom' is passed into the ConfGen.PyConSolv.run() function, a solvent parametrization interface is

started. Here, the user is required to input the absolute path to the location of the custom solvent XYZ file and subsequently provide information about the permittivity and refraction index of the solvent.

5.2. Keywords for the pyconsolv.restart file

Once PyConSolv is running, a pyconsolv.restart file is created in the folder where your input XYZ file is located. This file contains a keyword, which represents the last successful step of the parametrization. The values taken can be setup, orca, antechamber, frcmod, multiwfn, mcpb, tleap or equilibration. If you want to restart the parametrization at a specific point, you may edit this file and restart the PyConSolv process.