

PyConSolv 1.0.6

User Manual

Contents

1. Introduction.....	2
2. Installation.....	2
2.1. Software Requirements	2
2.2. Python Requirements.....	2
2.3. Setting up the python environment.....	2
3. Usage	3
3.1. Command line usage	3
3.2. Usage within python	4
4. Methodology	6
4.1. Program structure and files	6
4.2. Charge Fitting.....	7
4.3. MCPB.py	7
4.4. AmberMD	7
4.5. Solvents	8
5. Keywords.....	9
5.1. Keywords for PyConSolv	9
5.2. Keywords for the pyconsolv.restart file	9
6. Tutorial.....	10
6.1. Parametrization and simulation.....	10
6.2. Analysis and clustering.....	14

1. Introduction

PyConSolv is a python-based software package meant to simplify the setup for molecular dynamics (MD) simulations for transition metal containing catalysts, as well as analyse the resulting trajectories and generate conformers. 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](#)¹ (Tested with ORCA 5.0.4)
- [AmberTools](#)² (Tested with version 20-22)
- [Multiwfn](#)³ (Tested with version 3.8)

2.2. Python Requirements

To install PyConSolv on your system, Python 3.10 is needed, as well as the following requirements:

- numpy >= 1.23.1
- pandas >= 1.4.3
- matplotlib >= 3.5.3
- rdkit >= 2022.09.1
- parmed >= 4.2.0

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 snippet 1, below:

```
#For Conda installations:
```

```
conda create -c conda-forge --name PyConSolv python=3.10 rdkit numpy  
pandas parmed
```

```
conda activate PyConSolv
```

```
pip install PyConSolv
```

```
#For PIP environments:
```

```
python3 -m venv env
```

```
source env/bin/activate
```

```
pip install numpy pandas rdkit parmed PyConSolv
```

Code snippet 1 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 or a counterion which has not been pre-parametrized is required ([see solvent keyword](#)), an empty folder should be created and an XYZ structure of your desired structure should be placed inside. The recommended folder structure looks as shown in figure 1, below.

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

Figure 1 initial folder structure example

Once the desired folder structure is complete, PyConSolv can be started on the input structure.

3.1. Command line usage

The PyConSolv parametrization process can be started by calling the pyconsolv script. For the execution of the script an XYZ input file must be provided.

A number of optional parameters can be passed to the function. For details of the available keywords, please see the [keywords](#) 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 suitable 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 snippet 2.

```
pyconsolv path/to/input.xyz -c 0 -m BP86 -b def2-SVP -d D4 -s CH2Cl2
```

Code snippet 2 Simple example on how to run pyconsolv from the command line.

3.2. Usage within python

The ConfGen module must be imported from the PyConSolv package to make the parametrization functions available. 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 [keywords](#) 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 suitable 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 snippet 3.

```
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 3 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.

```

Pyconsolv_structure/
├── Structure/
│   ├── input.xyz
│   ├── input.xyz.original
│   ├── pyconsolv.restart
│   ├── MCPB_setup/
│   │   ├── SLV.frcmod
│   │   ├── SLV.mol2
│   │   ├── LIG_tleap.in
│   │   ├── filenames.restart
│   │   ├── metalConnections
│   │   ├── input.in
│   │   ├── MCPB_s1.out
│   │   ├── MCPB_s2.out
│   │   ├── MCPB_s4.out
│   │   ├── Connections
│   │   └── various input/output files for antechamber and MCPB.py
│   ├── equilibration/
│   │   ├── LIG_solv.prmtop
│   │   ├── 00.rst7
│   │   └── various input/output files for Amber
│   ├── orca_calculations/
│   │   ├── opt/
│   │   │   ├── orca_opt.in
│   │   │   ├── input.xyz
│   │   │   └── various orca output files
│   │   ├── freq/
│   │   │   ├── multiwfn.input
│   │   │   ├── orca_freq.in
│   │   │   ├── input.xyz
│   │   │   └── various orca output files
│   └── simulation/
│       ├── LIG_solv.prmtop
│       ├── simulation.in
│       └── eq_done.rst7
├── Solvent/
│   ├── solvent_input.xyz
│   ├── done
│   ├── solv_param/
│   │   ├── input.xyz
│   │   ├── SLV.frcmod
│   │   ├── SLV.mol2
│   │   ├── various input/output files for antechamber
│   │   └── solv_param/
│   │       ├── input.xyz
│   │       ├── orca_opt.inp
│   │       ├── multiwfn.input
│   │       └── various orca output files

```

Figure 2 Folder structure and important files after parametrization is complete

4. Methodology

4.1. Program structure and files

PyConsolv follow a workflow that consists of multiple steps:

1. Folder structure setup

During this step, the folder structure required for PyConSolv is created and the input file is checked for compatibility. If required, the input file is re-organized in such a way as to allow for an error-free parametrization, by grouping the ligand structures together. ORCA input files are also created.

2. Geometry optimization and frequency calculation

Structure optimization and a subsequent frequency calculation is performed using ORCA

3. Fragment splitting

The individual fragments are detected and split up in preparation for parametrization using a depth first search algorithm.

4. Fragment charge assignment

A UI window pops up with a Lewis structure representation of each individual substructure, where the user must provide a charge for the ligand/metal.

5. Generation of initial forcefield parameter files

Each ligand is parametrized using antechamber, with the previously provided total charge.

6. RESP charge calculation

Using Multiwfn 3.8, the RESP charges are [calculated](#).

7. MCPB.py execution

MCPB.py input file is created and the script is then executed, in order to determine the metal parameters.

8. Solvent box construction

A solvent box is constructed, using the desired solvent and counterion (if applicable).

9. Equilibration setup

An equilibration [workflow](#) is prepared, along with all the needed input files. Using the GPU accelerated implementation of pmemd (Amber), the system is then equilibrated.

10. Simulation setup

The equilibrated system is copied to the simulation folder and some files required for a final analysis are created. A script is provided for starting the simulation using pmemd (Amber)

11. Analysis

The analysis of a simulation can be performed by calling the `pyconsolv` script with the `-a` option and providing the simulation basename as input (e.g. “`pyconsolv sim-01 -a`”)

4.2. Charge Fitting

`Multiwfn 3.81` is used to compute the charges for the parametrizations, as follows.

Nuclear + Electronic Merz-Kollman RESP1 type atomic charge mode is used with 6 points/Å². All atoms are used for fitting 4 layers, with the scaling factors: 1.4,1.6,1.8,2.0. Automatic radii are utilized for fitting, missing radii are taken from UFF and scaled by 1/1.2. For fitting, the tightness parameter is 0.1, the restraint strengths for the 2 stages are set to 0.0005 and 0.0010, respectively. The maximum number of iterations is 50 and the convergence threshold is set to 10⁻⁶. Charge equivalence constraints are enabled for CH₂ and CH₃ groups. No charge constraint settings are used and the connectivity is determined automatically.

These settings can be changed by the end user, by stopping the parametrization and modifying the `multiwfn.input` file created in the orca frequency calculation folder, followed by re-running `pyconsolv`.

4.3. MCPB.py

The `MCPB.py` module of `AmberTools` is used in the parametrization process. With the help of it, the parameters for the metal center are created, using the Seminario method, by taking the force constants from the hessian calculated for the system at hand.

The end result is the creation of a force-field modification file and structure files which can be imported into `tleap` for the solvation of the structure.

4.4. AmberMD

By default, `PyConSolv` is built with Amber support in mind. Not only relying on the `MCPB.py` module from `AmberTools` for parametrization, but also on the Amber MD engine for equilibration and MD simulations.

Once the parametrization step is complete an automatic equilibration procedure is started, as detailed below:

1. The entire simulation box is restrained using cartesian restraints (restraint weight = 1000 kcal mol⁻¹ Å⁻²), with the exception of the hydrogen atoms, which undergo an energy minimization for 500 cycles.
2. The solute heavy atoms remain restrained and the whole system is minimized for 1000 cycles (restraint weight = 1000 kcal mol⁻¹ Å⁻²).
3. The system is equilibrated under constant volume to achieve a temperature of 300K, using Langevin dynamics with the collision frequency of 2.0 ps⁻¹, a timestep of 0.001 ps and the SHAKE algorithm for hydrogen bonds.
4. The solute heavy atoms remain restrained (restraint weight = 1000 kcal mol⁻¹ Å⁻²) and the whole system is equilibrated at constant pressure (1 atm), to achieve a reasonable system density and eliminate the voids created by the automated solvent placement in `tleap`. This is achieved using Langevin dynamics with the collision frequency of 2.0 ps⁻¹, a timestep of 0.001 ps and the SHAKE algorithm for hydrogen bonds.

5. The system is subsequently cooled under constant volume to achieve a temperature of 100K, using Langevin dynamics with the collision frequency of 2.0 ps^{-1} , a timestep of 0.001 ps and the SHAKE algorithm for hydrogen bonds.
6. A stepwise reduction of the restraints takes place, from 1000 to 0 $\text{kcal mol}^{-1} \text{ \AA}^{-2}$, minimizing at each step (1000, 500, 200, 100, 50, 20, 10, 5, 4, 3, 2, 1, 0.5 $\text{kcal mol}^{-1} \text{ \AA}^{-2}$)
7. The system is heated up under constant volume to achieve a temperature of 300K, using Langevin dynamics with the collision frequency of 2.0 ps^{-1} , a timestep of 0.001 ps and the SHAKE algorithm for hydrogen bonds.
8. A final constant pressure (1 atm) equilibration takes place, using Langevin dynamics with the collision frequency of 2.0 ps^{-1} , a timestep of 0.001 ps and the SHAKE algorithm for hydrogen bonds.

The equilibrated simulation box is taken as the starting point for the classical MD simulation. This takes place under constant pressure (1 atm, Berendsen barostat) with isotropic position scaling, a pressure relaxation time of 2.0 ps with Langevin dynamics, SHAKE for the hydrogen atoms and a timestep of 0.001 ps.

4.5. Clustering

Three clustering methods are currently supported: Kmeans, DBscan and Hierarchical. For each of these methods, the user is required to provide some input, such as number of clusters or thresholds for cluster determination. For the clustering, an interface to CPPTRAJ⁴ is called.

4.6. Solvents

The following solvents are natively supported by PyConSolv: Acetone, Acetonitrile, Ammonia, Benzene, Tetrachloromethane, Dichloromethane, Chloroform, Cyclohexane, DMF, DMSO, Ethanol, Hexane, Methanol, 1-Octanol, Pyridine, THF, Toluene, Water.

These (with the exception of water) have been parametrized for use with PyConSolv, and verified that they approach experimental densities.

5. Keywords

5.1. Keywords for PyConSolv

Parametrization options:

- **-c, charge** : charge of the complete system; type int; default 0
- **-m, method** : ORCA 5 method line; type string; default 'PBE0'
- **-b, basis** : Basis set for ORCA calculations; type string; default 'def2-SVP'
- **-d, dsp** : Dispersion corrections; type string; default 'D4'
- **-p, cpu** : number of CPU cores to be used; type int; default 12
- **-s, solvent** : solvent to be used for MD simulation; type string; default 'Water'.
- **-mult** : multiplicity of the system, default 1
- **-noopt** : perform a single point calculation instead of a geometry optimization
- **-box** : set the solvent box size, default 10
- **-e, --engine** : choose the simulation engine to run the equilibration and simulation with. Options are amber and gromacs, amber is default.
- **-rst, --restraint** : perform a restrained simulation

Analysis options:

- **-a, --analyze** : analyze a simulation
- **-mask, --mask** : atomid mask for clustering
- **-cluster, --cluster** : clustering method
- **-v, --version** : show program's version number and exit
- **-nosp** : skip single point calculations for clusters
- **-qmmm** : use a qmmm approach to determine cluster energy ranking

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 to pyconsolv, a solvent parametrization dialogue 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, equilibration or DONE. If you want to restart the parametrization at a specific point, you may edit this file and restart the PyConSolv process.

6. Tutorial

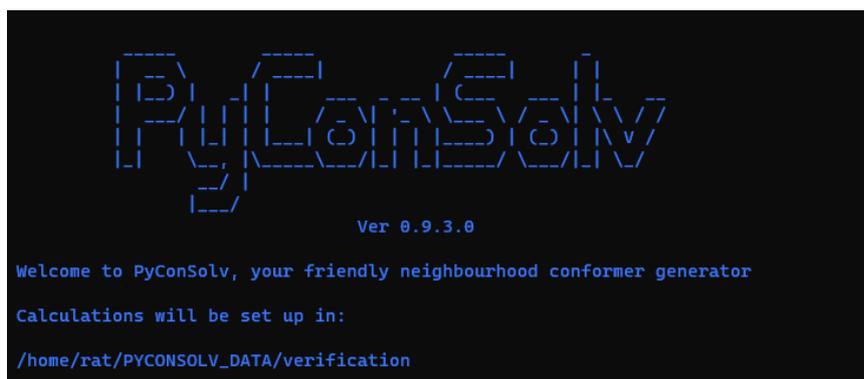
6.1. Parametrization and simulation

This is a basic tutorial for running PyConSolv from the command line.

The first step is to call the “pyconsolv” command with the appropriate parameters for an input file in an empty directory e.g.:

```
pyconsolv input.xyz -c 0 -m BP86 -b def2-SVP -d D4 -s Water
```

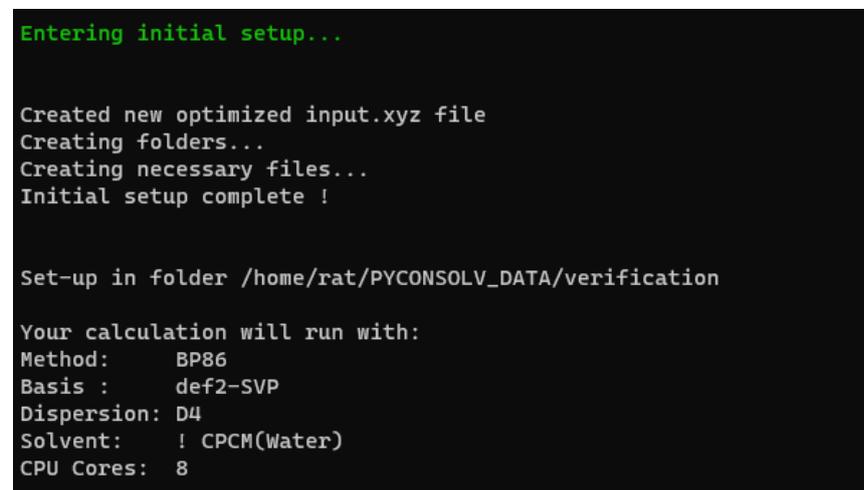
PyConSolv will show a greeting message and show the folder where the calculations will take place, as shown in figure 3, below.



```
PyConSolv
Ver 0.9.3.0
Welcome to PyConSolv, your friendly neighbourhood conformer generator
Calculations will be set up in:
/home/rat/PYCONSOLV_DATA/verification
```

Figure 3 PyConSolv greeting message and calculation folder location.

Following the greeting message, more information is provided, regarding the supplied input (see figure 4)



```
Entering initial setup...
Created new optimized input.xyz file
Creating folders...
Creating necessary files...
Initial setup complete !

Set-up in folder /home/rat/PYCONSOLV_DATA/verification

Your calculation will run with:
Method:      BP86
Basis :      def2-SVP
Dispersion:  D4
Solvent:     ! CPCM(Water)
CPU Cores:   8
```

Figure 4 Folder structure is created and the input options are reflected in the output message

Once setup is complete, a geometry optimization is performed on the input structure, followed by a frequency calculation (figure 5).

```
Setup is complete, moving on to ORCA calculations...

Found ORCA in: /usr/local/bin/orca_5.0.4
Running geometry optimization in /home/rat/PYCONSOLV_DATA/verification/orca_calculations/opt
/usr/local/bin/orca_5.0.4/orca orca_opt.inp > orca_opt.out

Calculation completed successfully!
Moving on!

Generating molden input file from calculation..

Reading the input file orca_opt.gbw          ... done
Writing the output file orca_opt.molden.input ... done
Running frequency calculation in /home/rat/PYCONSOLV_DATA/verification/orca_calculations/freq
/usr/local/bin/orca_5.0.4/orca orca_freq.inp > orca_freq.out

Calculation completed successfully!
Moving on!
```

Figure 7 ORCA calculations are performed automatically.

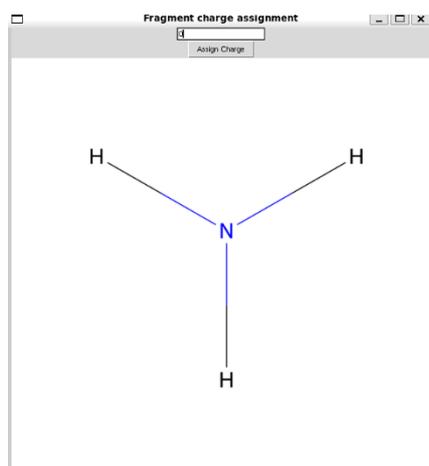


Figure 6 Charge assignment dialogue. The user is asked to input the appropriate charge for each fragment.

```
Running antechamber for B.pdb
Antechamber completed successfully for B.pdb

Running antechamber for C.pdb
Antechamber completed successfully for C.pdb

Generating frcmod files for ligands:

Generated B.frcmod...
Generated C.frcmod...
Generated D.frcmod...
Generated E.frcmod...
```

Figure 5 The charges are assigned to each fragment and the program proceeds to the generation of frcmod files.

Following the frequency calculation, a pop-up window is displayed to the user, asking for charge assignment for each fragment (figure 6).

Once the fragment charges have been assigned, a charge map file is written in the [MCPB_setup](#) folder for future reference (figure 7).

```
Generating molden input file from calculation..
Reading the input file orca_freq.gbwn           ... done
Writing the output file orca_freq.molden.input  ... done
ORCA calculations completed successfully!

ORCA Calculations complete, moving on to MCPB setup...
The following pdb files were created:
  PT.pdb B.pdb C.pdb D.pdb E.pdb

Please enter fragment charges:

Charges have been mapped to fragments as follows:

PT -> 2.0
B  -> 0.0
C  -> 0.0
D  -> -1.0
E  -> -1.0

Map written to /home/rat/PYCONSOLV_DATA/verification/MCPB_setup/chargeMap.dat
Reading charges from /home/rat/PYCONSOLV_DATA/verification/MCPB_setup/chargeMap.dat
```

Figure 8 Force field modification files are created.

Once the charge maps file was written, the force field modification files (frcmmod files) are created for each fragment, and the MCPB.py input file is created.

```
Fragments have been prepared, running MultiWfn task...

ECPs found, correcting molden input file

Atoms with ECP:
Pt with 18 electrons

RESP charge calculation completed successfully!
```

Figure 9 RESP charge calculation, preceeded by an ECP presence check in the molden file.

Once the frcmmod files have been created, the RESP charge calculations are started. This is performed with Multiwfn, which requires a molden formatted file as input. This is created automatically from the orca output, followed by a check for ECPs, which have to be marked for correct processing with Multiwfn (figure 9).

```

Converting ORCA output to MCPB.py compatible input...

Proceeding with MCPB steps...

MCPB.py step 1 completed successfully

Not all connections were autodetected by MCPB.py

Adding : 1-10

Adding : 1-11

MCPB.py step 1 completed successfully

```

Figure 10 MCPB.py parametrization is performed.

With the RESP charge calculation complete, the MCPB.py parametrization is started and the MCPB.py generated tleap file is corrected for missing bond and correctness, followed by the modification for the appropriate solvent and box size (figure 10). The box size is chosen automatically, such that the closest box edge is 15 Å away from the solute. This is adjusted automatically, in case the box is still too small, during the equilibration.

```

Setting up system equilibration...

Writing input files in the equilibration folder...

Done!

Starting equilibration...
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_OVERFLOW_FLAG IEEE_UNDERFLOW_FLAG IEEE_DENORMAL
Equilibration step 1 completed successfully

```

Figure 11 System equilibration is started.

The equilibration is performed and the user is informed of the success at the end of the equilibration step (figure 11). Here the next suggested steps are provided to the user (figure 12)

```

Preparing simulation...

Simulation setup complete, please execute the run_simulation.sh script in:
/home/rat/PYCONSOLV_DATA/verification/simulation
to begin a 100ns cmd production run.

A quick analysis of the simulation run can be performed using the "pyconsolv sim-01 -a" command, in your simulation folder

My job here is done!

```

Figure 12 Equilibration is completed and next steps are suggested.

6.2. Analysis and clustering

The clustering module of PyConSolv can be called with the “-a” option on any simulation. To begin the analysis, the basename of the simulation file must be provided. The default basename for a simulation started with PyConSolv is sim-01. In this case, the analysis can be started with:

```
pyconsolv sim-01 -a
```

The user is asked to provide atom indexes for aligning the simulation (figure 13), followed by options regarding the clustering method.

```
Warning, you have not provided an input mask for alignment, please provide a list of atom ids in the format: 1,2,3-10
1,2-4
Please provide a clustering method:
kmeans
clustering using kmeans
Please enter the number of desired clusters: (default 10)
10
Please enter the value for the frame number sieve: (default 10)
1
Clustering done!
```

Figure 13 Interactive dialogue for the analysis and clustering of a simulation.

Once the clustering is done, single point energy calculations are automatically performed, at the same level of theory as the one used for parametrization. Finally, the energetic ranking of the conformers is printed out (figure 14), as well as saved in the “cluster_ranking.dat” output file.

```
Cluster Energy(Ha)
rep.c1 -1152.82258903024
rep.c6 -1152.82228905294
rep.c8 -1152.820944562936
rep.c7 -1152.820583488084
rep.c3 -1152.820059685063
rep.c0 -1152.816423507697
rep.c4 -1152.814920647322
rep.c5 -1152.812858320153
rep.c2 -1152.811893209523
rep.c9 -1152.807549712199
```

Figure 14 Cluster energy ranking using DFT single point energies.