Tutorial for 1-body additive QM/MM crystal optimization packages

Developed by Yalun Zheng

22. Jul. 2019

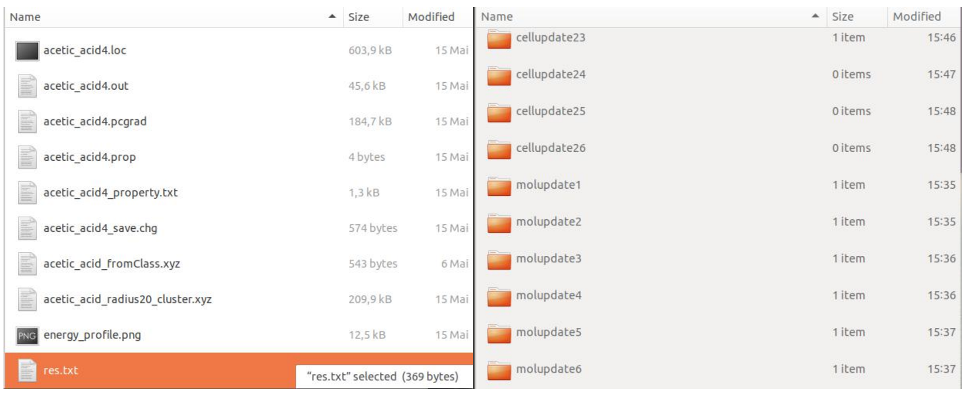
Contents

1. Scripts in Program

In the directory “crystaloptimizer”, there are many files, not only in the format of .py, but also some .txt files.
Let’s check them out!
acetic_acid.cif: an .cif example file for you to ensure you can immediately optimize an acetic acid. Sure, if you want, you can also download some other .cif files to test this program.
atom.py: definition of the atom class with atomic properties (label, type, coordinate, charge, etc.) and operations (translation, distance measurement to another atom, rotation, etc.).
combined_as_tt.py: the main script for the Lennard-Jones potential based optimization.
combined_new.py: the main script for the Buckingham potential based optimization.
execute.py: an interface to write an input file for QM packages and run QM calculations.
gradient.py: some functions and parameters for calculating the Buckingham potential gradient as well as the total gradient for all atoms.
gradient2.py: some functions and parameters for calculating the Lennard-Jones potential gradient as well as the total gradient for all atoms.
molecule.py: definition of molecule class with molecular properties and operations, also the calculation of rotation matrix between 2 molecules included here.
qmmm.dat: an example .dat file, where all needed information for an optimization should be written.
README.txt: a very simple introduction to how to run the program.
rfotest.py: the function for computing the RFO step vector.
stressastt.py: some functions for calculating the Buckingham potential based total stress of the crystal.
stressastt2.py: some functions for calculating the Lennard-Jones potential based total stress of the crystal.
toolkit.py: lots of used or unused tool-functions. The dictionary of rotational and translational matrixes of different space groups included as well.
top.py: the top level script to read .dat file and run all the needed functions during optimization.
Another 3 *_x.py files are written for developer.

2. Output Files

Program will generate many files and directories after optimizing the crystal.



Name	Size	Modified
acetic_acid4.loc	603,9 kB	15 Mai
acetic_acid4.out	45,6 kB	15 Mai
acetic_acid4.pcgrad	184,7 kB	15 Mai
acetic_acid4.prop	4 bytes	15 Mai
acetic_acid4_property.txt	1,3 kB	15 Mai
acetic_acid4_save.chg	574 bytes	15 Mai
acetic_acid_fromClass.xyz	543 bytes	6 Mai
acetic_acid_radius20_cluster.xyz	209,9 kB	15 Mai
energy_profile.png	12,5 kB	15 Mai
res.txt		

“res.txt” selected (369 bytes)

Name	Size	Modified
cellupdate23	1 item	15:46
cellupdate24	0 items	15:47
cellupdate25	0 items	15:48
cellupdate26	0 items	15:48
molupdate1	1 item	15:35
molupdate2	1 item	15:35
molupdate3	1 item	15:36
molupdate4	1 item	15:36
molupdate5	1 item	15:37
molupdate6	1 item	15:37

If you do not care about the optimization process, then **res.txt** includes all information you may need:

E: crystal energy,

V0: cell volume corresponding to .cif file,

V: cell volume corresponding to the optimization results (the following cell parameters),

a, b, c, alpha, beta, gamma: optimized cell parameters.

The **energy_profile.png** records the energy descent.

***_radius_20_cluster.xyz** includes all optimized atomic coordinates in crystal.

20energy.npy can be loaded by numpy and records the energy values after each step of optimization (value 0, 1, 2 are the same initial energy, please use [2:] to plot a similar figure to energy_profile.png).

20finalcluster.npy can also be loaded by numpy and records the optimized molecules in crystal in a form of list.

***.chg** saves all the surrounding atomic charges.

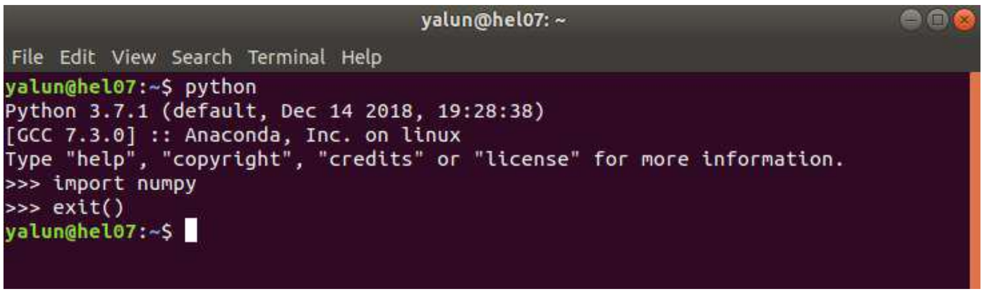
*.xyz saves the initial atomic coordinates of the central molecule according to the .cif file.

All other files are automatically generated during QM calculations. Inside of their file names, there is a number which represents iteration times of calculating atomic charges.

3. For Common Users

Here are steps to run the program.

1. Please make sure that python3 and numpy library are alrerady installed on your computer or on the server.



```
yalun@hel07:~$ python
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> exit()
yalun@hel07:~$
```

If there is no error after run “import numpy” and the version of python is 3.x.x. Then you can continue. (There might some users who can run “python3” alternatively. Don’t worry, it works as well.)

2. Now, you can run the program. Let’s do it using acetic acid as an example.

Run the top.py inside of the crystaloptimizer directory.

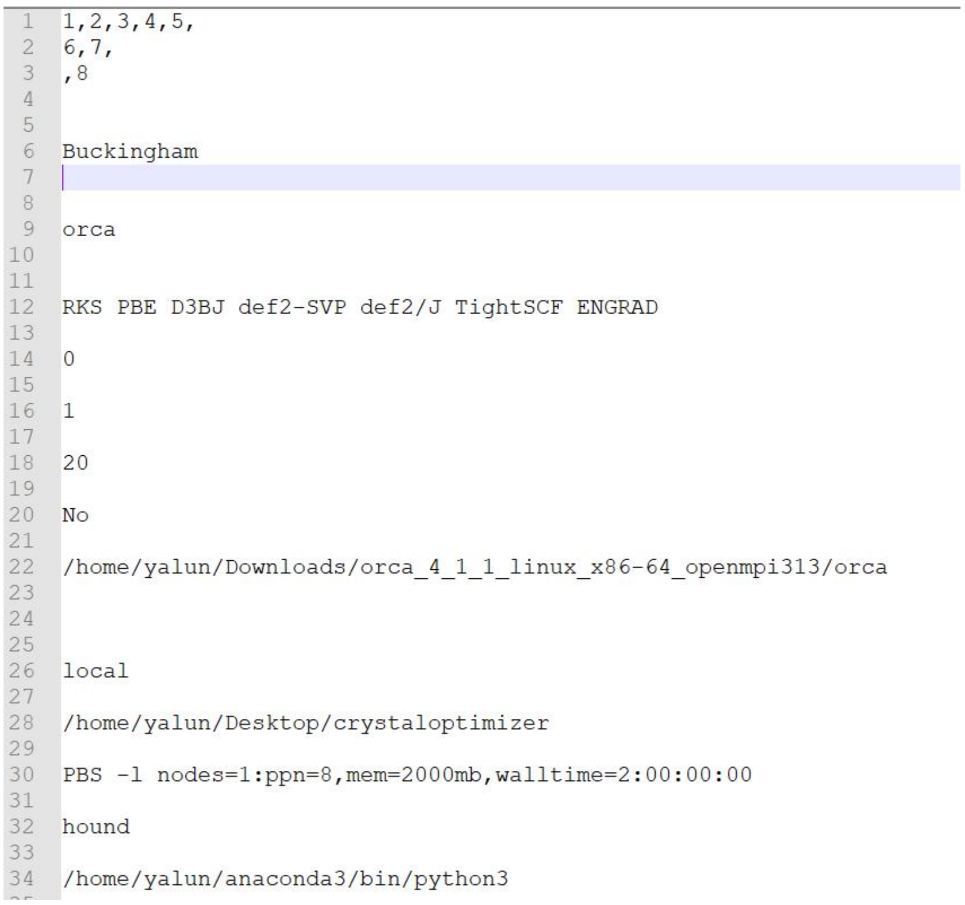
Create a root directory where you want to store all the generating files.

Input the path of the root directory.

Input the path of the .cif file of the crystal you want to optimize.

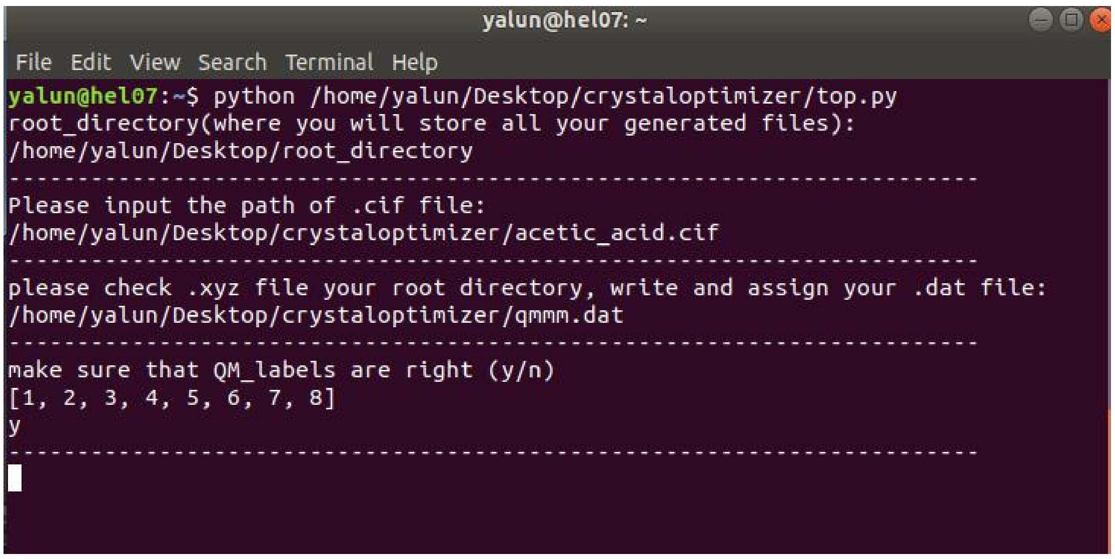
Now in the root directory, there should be a .xyz file. This file includes all atomic coordinate in 8 neighboring unit cells. Open it and choose the atom lines (e.g. 1, 2, 3, 4, 5, 6, 7, 8) which will be computed as QM molecules. The selected labels will be written in the .dat file*.

Write .dat file and input it into the terminal.



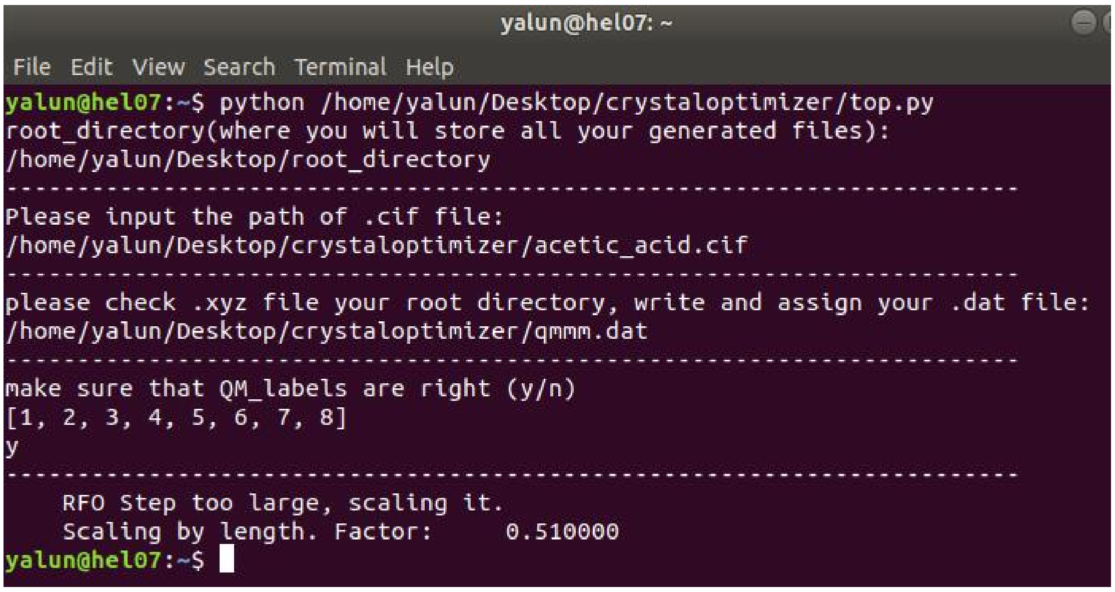
```
1,2,3,4,5,
6,7,
,8


Buckingham


orca


RKS PBE D3BJ def2-SVP def2/J TightSCF ENGRAD

0

1

20

No

/home/yalun/Downloads/orca_4_1_1_linux_x86-64_openmpi313/orca



local

/home/yalun/Desktop/crystaloptimizer

PBS -l nodes=1:ppn=8,mem=2000mb,walltime=2:00:00:00

hound

/home/yalun/anaconda3/bin/python3
```

The whole process is shown in the following.



```
yalun@hel07:~$ python /home/yalun/Desktop/crystaloptimizer/top.py
root_directory(where you will store all your generated files):
/home/yalun/Desktop/root_directory
-----------------------------------------------------------------------
Please input the path of .cif file:
/home/yalun/Desktop/crystaloptimizer/acetic_acid.cif
-----------------------------------------------------------------------
please check .xyz file your root directory, write and assign your .dat file:
/home/yalun/Desktop/crystaloptimizer/qmmm.dat
-----------------------------------------------------------------------
make sure that QM_labels are right (y/n)
[1, 2, 3, 4, 5, 6, 7, 8]
y
-----------------------------------------------------------------------
```

Don’t close this terminal until the cursor is released like the following screenshot.

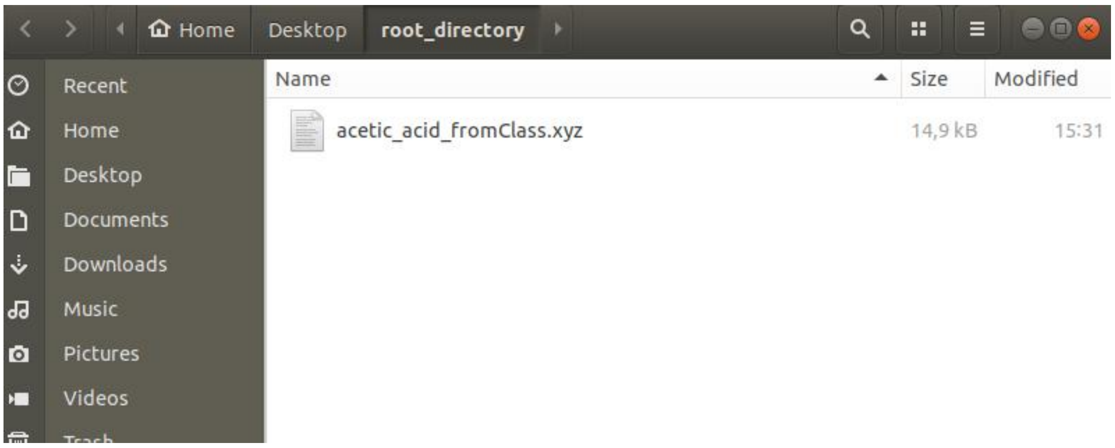


```
yalun@hel07:~$ python /home/yalun/Desktop/crystaloptimizer/top.py
root_directory(where you will store all your generated files):
/home/yalun/Desktop/root_directory
-----------------------------------------------------------------------
Please input the path of .cif file:
/home/yalun/Desktop/crystaloptimizer/acetic_acid.cif
-----------------------------------------------------------------------
please check .xyz file your root directory, write and assign your .dat file:
/home/yalun/Desktop/crystaloptimizer/qmmm.dat
-----------------------------------------------------------------------
make sure that QM_labels are right (y/n)
[1, 2, 3, 4, 5, 6, 7, 8]
y
-----------------------------------------------------------------------
    RFO Step too large, scaling it.
    Scaling by length. Factor:     0.510000
yalun@hel07:~$ 
```

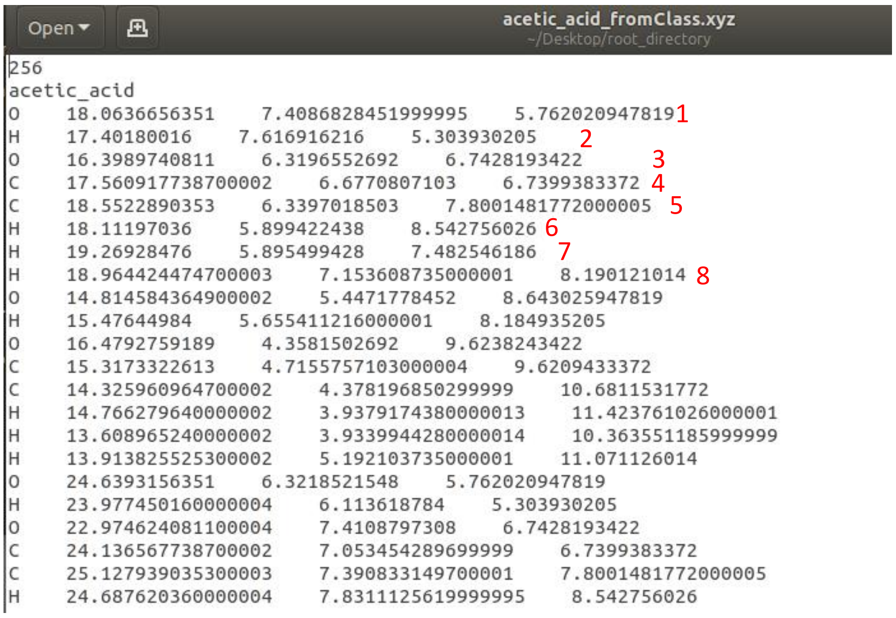
*About the .dat file

There are totally 14 blocks, from top to bottom they are:

1. **QM_labels block**: selected QM atom rows from the generated .xyz file in root directory you afforded before, starting from 1.



```
acetic_acid_fromClass.xyz    14,9 kB    15:31
```



```
256
acetic_acid
O    18.0636656351    7.4086828451999995    5.762020947819 1
H    17.40180016    7.616916216    5.303930205    2
O    16.3989740811    6.3196552692    6.7428193422        3
C    17.560917738700002    6.6770807103    6.7399383372 4
C    18.5522890353    6.3397018503    7.8001481772000005  5
H    18.11197036    5.899422438    8.542756026 6
H    19.26928476    5.895499428    7.482546186  7
H    18.964424474700003    7.153608735000001    8.190121014 8
O    14.814584364900002    5.4471778452    8.643025947819
H    15.47644984    5.655411216000001    8.184935205
O    16.4792759189    4.3581502692    9.6238243422
C    15.3173322613    4.7155757103000004    9.6209433372
C    14.325960964700002    4.378196850299999    10.6811531772
H    14.766279640000002    3.9379174380000013    11.423761026000001
H    13.608965240000002    3.9339944280000014    10.363551185999999
H    13.913825525300002    5.192103735000001    11.071126014
O    24.6393156351    6.3218521548    5.762020947819
H    23.977450160000004    6.113618784    5.303930205
O    22.974624081100004    7.4108797308    6.7428193422
C    24.136567738700002    7.053454289699999    6.7399383372
C    25.127939035300003    7.390833149700001    7.8001481772000005
H    24.687620360000004    7.8311125619999995    8.542756026
```

There must be at least 1 comma in one QM_labels line.

1,2,3,4,	
5,6	Perfect!
1,	Perfect!
8	Not acceptable!

2. **MM_model block**: Lennard-Jones(recommended) or Buckingham.

Lennard-Jones	Perfect!
lennard-Jones	Perfect!
Lennard-buckingham	Unknown potentials not acceptable

3. **QM_program block**: orca or Gaussian(did not finish).

Orca	Perfect!

4. **QM_command block**: do not need to write "opt" here, even though you will set opt to be Yes!

RKS PBE D3BJ def2-SVP def2/J TightSCF ENGRAD	Perfect!

5. **QM_charge block**: net charge of the QM molecule (generally 0 for molecular crystal).

0	Perfect!

6. QM_Spin block: spin multiplicity of the QM molecule (generally 1 for molecular crystal without unpaired electron).

1	Perfect!

7. **cluster_radius block**: usually 20 angstroms.

20	Perfect!

8. **optimization for gaseous molecule block**: optimize the central molecule and obtain its QM energy (Yes or No). Usually No, but when you really would like to know this energy of a gaseous molecule and do not want to optimize the crystal, set it to be Yes.

No	Perfect!

9. **QM packages path bock**: where did you uncompress QM packages locally. This block is valid only when you set the next block to be "local".

/home/yalun/Downloads/orca_4_1_1_linux_x86-64_openmpi313/orca	Perfect!

10. **run locally or on the server**:local or server.

local	Perfect!
Server	Perfect!

11. **optimizer_path**: where did you put my program.

/home/yalun/Desktop/crystaloptimizer	Perfect!

12. **shell_PBS**: Valid only when block 10 is set to be "Server".

PBS -l nodes=1:ppn=8,mem=2000mb,walltime=2:00:00:00	Perfect!

13. **queue block**: fish or elephant or hound. Calculation queue on the server. Valid only when block 10 is set to be "Server".

hound	Perfect!

14. **python_path block**: which python you would like to use during optimization. Only valid when block 10 is set to be "Server".

/home/yalun/anaconda3/bin/python3	Perfect!

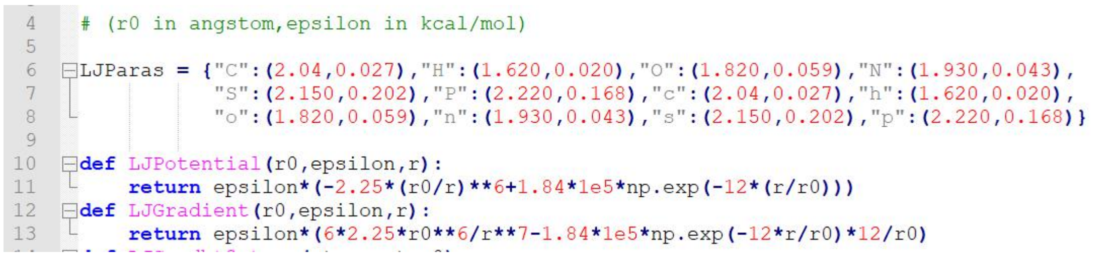
4. For Developer

If you would like to use Gaussian, then I can not help you by this Tutorial. If you have any question about my thoughts or my coding, ask me by email: 1799253039@qq.com

If you would like to try other charge distributions or vdw potential functions, I can give you guidance here.

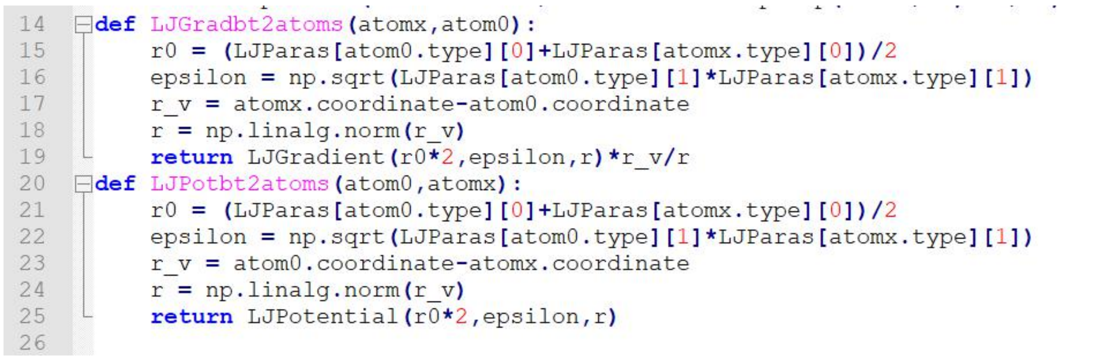
4.1. Try Another Vdw Porential Function

1. In the dat file, you must set block 2, as well as the MM_model block, to be potential_x.
2. In the directory named crystaloptimizer, I have written 2 extra .py files, gradient_x.py and stressastt_x.py. What you should do is just to open gradient_x.py and change LJParas, LJPotential and LJGradient to be what ever you want to test. Do not change their names please. Just substitute parameters and the potential function as well as its derivative.



```python
# (r0 in angstom,epsilon in kcal/mol)

LJParas = {"C":(2.04,0.027),"H":(1.620,0.020),"O":(1.820,0.059),"N":(1.930,0.043),
           "S":(2.150,0.202),"P":(2.220,0.168),"c":(2.04,0.027),"h":(1.620,0.020),
           "o":(1.820,0.059),"n":(1.930,0.043),"s":(2.150,0.202),"p":(2.220,0.168)}

def LJPotential(r0,epsilon,r):
    return epsilon*(-2.25*(r0/r)**6+1.84*1e5*np.exp(-12*(r/r0)))
def LJGradient(r0,epsilon,r):
    return epsilon*(6*2.25*r0**6/r**7-1.84*1e5*np.exp(-12*r/r0)*12/r0)
```

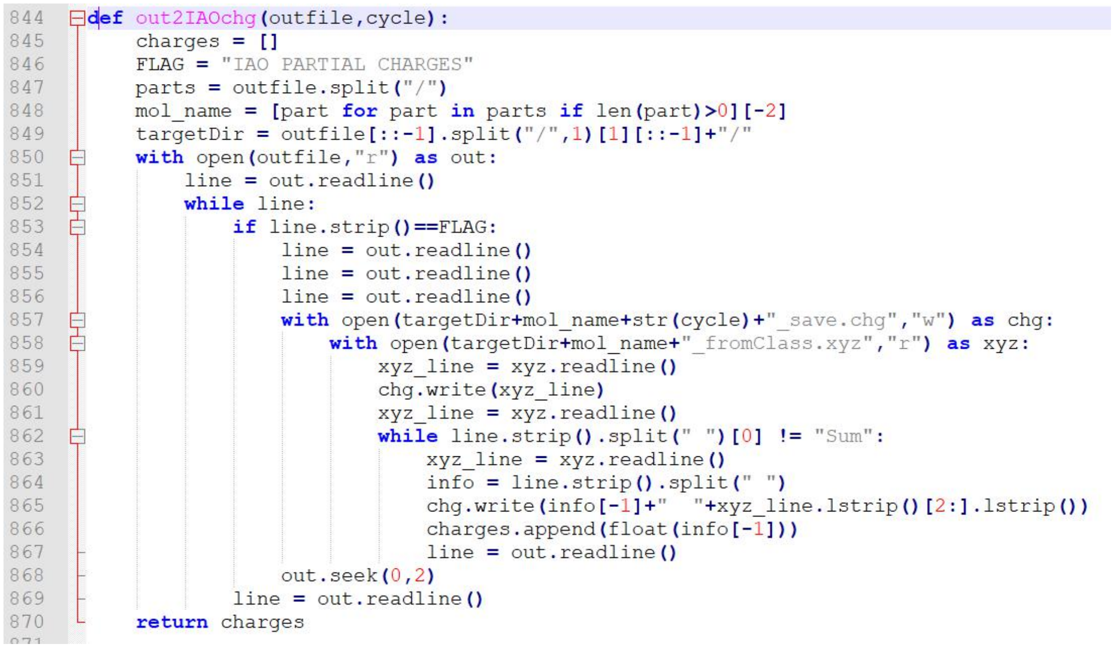
3. Change the radius rule (arithmetic or geometric mean) and energy rule for functions LJGradbt2atoms and LJPotbt2atoms, if you need.



```python
def LJGradbt2atoms(atomx,atom0):
    r0 = (LJParas[atom0.type][0]+LJParas[atomx.type][0])/2
    epsilon = np.sqrt(LJParas[atom0.type][1]*LJParas[atomx.type][1])
    r_v = atomx.coordinate-atom0.coordinate
    r = np.linalg.norm(r_v)
    return LJGradient(r0*2,epsilon,r)*r_v/r
def LJPotbt2atoms(atom0,atomx):
    r0 = (LJParas[atom0.type][0]+LJParas[atomx.type][0])/2
    epsilon = np.sqrt(LJParas[atom0.type][1]*LJParas[atomx.type][1])
    r_v = atom0.coordinate-atomx.coordinate
    r = np.linalg.norm(r_v)
    return LJPotential(r0*2,epsilon,r)
```

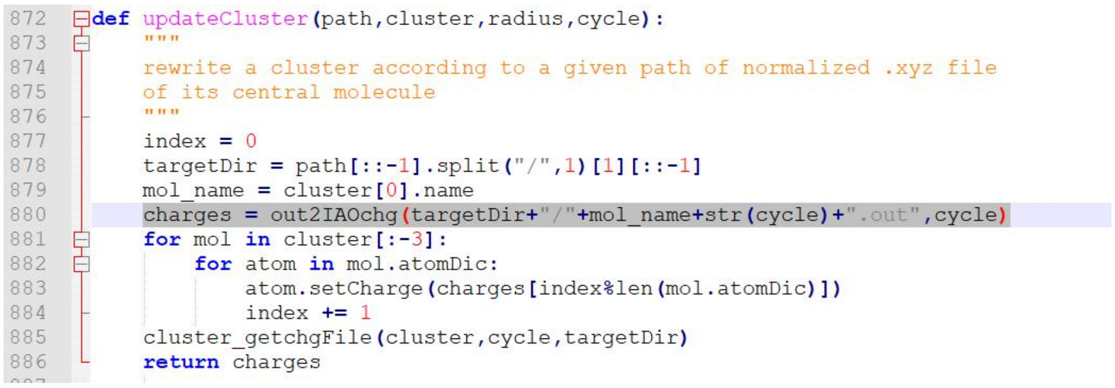
4.2. Try Another Charge Model

1. In the data file, you should rightly write a QM command and ensure to generate that kind of charges you need in .out file.
2. Open toolkit.py and find the function out2IAOchg. This is a function I wrote to search for IAO charges from the .out file. You can also write a similar function and return charges you want as a list.



```python
def out2IAOchg(outfile,cycle):
    charges = []
    FLAG = "IAO PARTIAL CHARGES"
    parts = outfile.split("/")
    mol_name = [part for part in parts if len(part)>0][-2]
    targetDir = outfile[::-1].split("/",1)[1][::-1]+"/"
    with open(outfile,"r") as out:
        line = out.readline()
        while line:
            if line.strip()==FLAG:
                line = out.readline()
                line = out.readline()
                line = out.readline()
                with open(targetDir+mol_name+str(cycle)+"_save.chg","w") as chg:
                    with open(targetDir+mol_name+"_fromClass.xyz","r") as xyz:
                        xyz_line = xyz.readline()
                        chg.write(xyz_line)
                        xyz_line = xyz.readline()
                        while line.strip().split(" ")[0] != "Sum":
                            xyz_line = xyz.readline()
                            info = line.strip().split(" ")
                            chg.write(info[-1]+"   "+xyz_line.lstrip()[2:].lstrip())
                            charges.append(float(info[-1]))
                            line = out.readline()
                out.seek(0,2)
            line = out.readline()
    return charges
```

3. Also in toolkit.py, change the line 880, "charges = out2IAOchg(targetDir+"/"+mol_name+str(cycle)+".out",cycle)", to be "charges = ****(targetDir+"/"+mol_name+str(cycle)+".out",cycle)". **** is the name of the function you wrote in step 2.



```python
872  def updateCluster(path,cluster,radius,cycle):
873      """
874      rewrite a cluster according to a given path of normalized .xyz file
875      of its central molecule
876      """
877      index = 0
878      targetDir = path[::-1].split("/",1)[1][::-1]
879      mol_name = cluster[0].name
880      charges = out2IAOchg(targetDir+"/"+mol_name+str(cycle)+".out",cycle)
881      for mol in cluster[:-3]:
882          for atom in mol.atomDic:
883              atom.setCharge(charges[index%len(mol.atomDic)])
884              index += 1
885      cluster_getchgFile(cluster,cycle,targetDir)
886      return charges
```