Computational Chemistry Laboratory III (CBE 60547)

Prateek Mehta, William F. Schneider

03-05-2015 Thu

1 A review of what we know

So far we have learned how to:

- navigate the linux terminal
- create create and edit files using Emacs
- numerical analysis and plotting with Python
- different concepts in molecular simulations, e.g. potential energy surfaces, geometry optimizations, etc.

It might make sense to go back and read the lecture notes and notes from Lab 1 and Lab 2 if you feel the need to re-familiarize yourself with these things. In this lab, we will combine some of the things we learned and to perform DFT calculations with a powerful software package, `VASP`.

2 Loading the required software

Before we can actually proceed, we will need to tell our computer how it can find all the tools we need. We will store this information so that the software is already loaded for us every time we login in the future. Depending on your unix shell, the things we need to do will be a little different. You can see which shell you are using with the command `echo $0`.

The software we need is dependent on the bash shell and thus we (all users) need to add a few commands to our `.bashrc` file. Go to your home directory and open the `.bashrc` file, i.e., run the following two commands.

```
cd
emacs .bashrc
```

Once there, add the following line at the bottom of the file and save it (Make sure there are no typos!).

```
source /afs/crc.nd.edu/user/w/wschnei1/CBE547/software/course_bashrc.sh
```

If there is a line that says `module load ase` in your file, remove it.

For tsch users, there is an additional step. Go to the terminal and run,

```
cd
emacs .cshrc
```

Once there, add the following line at the bottom of the file and save it (Make sure there are no typos!).

`source /afs/crc.nd.edu/user/w/wschnei1/CBE547/software/course_cshrc.sh`

If there is a line that says `module load ase` in your file, remove it.

Now logout and log back in. Once this is done, go to computational-chemistry/Lab3/ and open it Lab3.org in emacs.

3 Introduction to Software

`VASP` or the Vienna ab-inito Simulation Package is a density functional theory (DFT) package that utilizes periodic boundary conditions and planewave basis sets. It was developed at the Theoretical Physics Department at the Institute for Materials Physics in Vienna, Austria. More information about `VASP` can be obtained at http://cms.mpi.univie.ac.at/vasp/vasp/vasp.html. We will use a combination of the `ASE` (https://wiki.fysik.dtu.dk/ase/index.html) and `jasp` (https://github.com/jkitchin/jasp) packages to help prepare input files, manage job submission to the queue system, and analysis of results.

Note: The original `jasp` code has been slightly modified to work with the Notre Dame queue system.

Prof. J. R. Kitchin wrote a book to accompany `jasp` (http://kitchingroup.cheme.cmu.edu/dft-book/). It contains 100s of examples of using `jasp` for almost every kind of calculation that can be performed using `VASP`. Most of the examples in this document were from that book!

Note: This is not the most recent version of the book, so some of the functionality might be different to reflect the most recent version of `ase`. For the most recent version go to, https://github.com/jkitchin/dft-book.

4 Creating Molecules

Molecules are defined in `ase` using something called Atoms objects, which are a combination of Atom objects (obviously!). There are various ways to create Atoms objects - by hand, reading them from files, databases, etc.

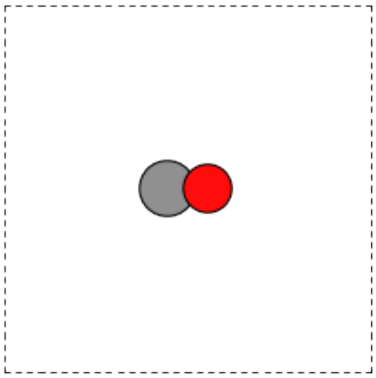
4.1 From Scratch

We can build atoms by hand by specifying the type and position of each atom, and the unit cell the atoms are in.

```
from ase import Atoms, Atom
from ase.io import write
from ase.visualize import view
```

```
4
5  # define an Atoms object
6  atoms = Atoms([Atom('C', [0., 0., 0.]),
7                 Atom('O', [1.1, 0., 0.])],
8                cell=(10, 10, 10))
9
10 atoms.center() # For a better visualization
11
12 print('V = {0:1.0f} Angstrom^3'.format(atoms.get_volume()))
13 write('images/simple-cubic-cell.png', atoms, show_unit_cell=2)
14 view(atoms)
```

```
V = 1000 Angstrom^3
```



4.2 Using in-built databases

We can load predefined molecules from `ase.structure.molecule`. For example, the database contains the molecules in the G2 set (http://www.cse.anl.gov/OldCHMwebsiteContent/compmat/comptherm.htm) among others. These are generally the result of MP2/6-31g(d) calculations from a code like `GAUSSIAN` or `GAMESS`. Consequently, they will not have unit cell information, and will have a default unit cell of ((1. 0. 0.), (0. 1. 0.), (0. 0. 1.)). We need to manually specify the unit cell for a `VASP` calculation.

```
1  from ase.structure import molecule
2  from ase.visualize import view
3
4  atoms = molecule('CO')
5
6  view(atoms)
7  print atoms
8  print 'Old Cell:'
9  print atoms.get_cell()
10
11 atoms.set_cell((10,10,10), scale_atoms=False)
12 print 'New Cell:'
13 print atoms.get_cell()
14 view(atoms)
```

The g2 set as implemented in ase is given below.

`isobutene`	`CH3CH2OH`	`CH3COOH`
`COF2`	`CH3NO2`	`CF3CN`
`CH3OH`	`CCH`	`CH3CH2NH2`

PH3
O2
Be
C3H9N
CH3
C2H6CHOH
CH3O
P
OH
SiCl4
C3H7
CH2NHCH2
SiF4
OCS
SH2
CH3CH2Cl
HCOOH
2-butyne
HOCl
C
C4H4S
H2CCHCl
C2H4
C2Cl4
C6H6
H2CCHCN
HCl
Na
CH3CH2SH
C2H5
ClO
NO
HCO
CH3COCH3
CO
CH3COCl
Si
N2
F2
CH2OCH2
SO
ClF3
B
C3H6_Cs
LiF
LiH
C2H6SO
Na2
Si2H6
BCl3
H2CCl2
CH3CH2OCH3
CH4
SiH2_s1A1d
H
C3H4_C3v
methylenecyclopropane
HCF3
CH3CH2O
SiH2_s3B1d
H2CCO
HF
C3H4_C2v
isobutane
CH3ONO
SH
CS2
CH3S
S
C2H6
HCN
bicyclobutane
N2H4
H2CCHF
CH3OCH3
CH3SiH3
OCHCHO
SiH3
AlCl3
C2H3
CH3CONH2
C3H4_D2d
CN
N
C3H8
Cl2
CO2
H2O
HCOOCH3
Li
CH3SH
C2H6NH
H2COH
SiO
C5H5N
C4H4O
O3
CH2_s1A1d
C3H9C
BF3
S2
H3CNH2
BeH
C2F4
F2O
HCCl3
AlF3
H2CF2
PH2
NO2
H2O2
CH3COF
C5H8
NF3
P2
O
C3H7Cl
CH3CHO
C2H2
H2
C4H4NH
cyclobutane
Li2
NaCl
SiH4
NH
CCl4
ClF
CH2SCH2
CH
F
CH3Cl
CS
NCCN
Cl
CH3CO
butadiene
PF3
CF4
N2O
cyclobutene
Si2
trans-butane
SO2

```
NH3                 NH2                 CH2_s3B1d
ClNO                C3H6_D3h            Al
CH3SCH3             H2CO                CH3CN
```

4.3 Reading structures from files

ASE can read a variety of data formats using `ase.io.read`. For example, here is a cif file I downloaded from http://materialsproject.org.

mp-22862_NaCl.cif

```
from ase.io import read
from ase.visualize import view

atoms = read('mp-22862_NaCl.cif')

view(atoms)
print atoms
```

```
Atoms(symbols='Na4Cl4', positions=..., cell=[[5.69169356, 0.0, 0.0], [3.485157149990802e-16, 5
```

5 Simple SCF calculations

We will now perform a simple calculation on our CO molecule. This is done by creating a `jasp` calculator, which is an extension of the default Vasp calculator in ase (`ase.calculators.vasp`). The two properties that we will calculate in this example are the energy and the forces on the atoms.

The first time we run this code, a calculation will be submitted to the Notre Dame queue system. So when you try to print the potential energy of you will get an exception saying VaspSubmitted. You can check the status of the job by going back to the terminal and typing `qstat -u netid`. Once the job has finished running you can rerun the code, and if all went well, it should give you the energies and the forces.

```
from ase import Atoms, Atom
from ase.io import write
from ase.visualize import view
from jasp import jasp

JASPRC['queue.q'] = 'long'
JASPRC['queue.nprocs'] = 8
JASPRC['queue.pe'] = '*@@schneider'

# define an Atoms object
co = Atoms([Atom('C', [0., 0., 0.]),
            Atom('O', [1.1, 0., 0.])],
           cell=(10, 10, 10))

with jasp('molecules/simple-co',  # output dir relative to current dir
          xc='PBE',  # the exchange-correlation functional
          nbands=8,    # number of bands
          encut=350,    # planewave cutoff
          ismear=1,    # Methfessel-Paxton smearing
          sigma=0.01,  # very small smearing factor for a molecule
          atoms=co) as calc:
```

```
22      print('energy = {0} eV'.format(co.get_potential_energy()))
23      print 'Forces (eV/Ang.):'
24      print(co.get_forces())
25      print 'SCF iterations = {0}'.format(calc.get_number_of_iterations())
26      print calc # Prints a summary of the calculation
27      #Note: some properties are attributes of the atoms object and some of the calc.
```

```
energy = -14.69232797 eV
Forces (eV/Ang.):
[[-5.777  0.     0.   ]
 [ 5.777  0.     0.   ]]
SCF iterations = 16
: ------------------------------
  VASP calculation from /afs/crc.nd.edu/user/p/pmehta1/computational-chemistry/Lab3/molecules/s
  converged: True
  Energy = -14.692328 eV

  Unit cell vectors (angstroms)
        x       y     z      length
  a0 [ 10.000  0.000  0.000] 10.000
  a1 [ 0.000  10.000  0.000] 10.000
  a2 [ 0.000  0.000  10.000] 10.000
  a,b,c,alpha,beta,gamma (deg):10.000 10.000 10.000 90.0 90.0 90.0
  Unit cell volume = 1000.000 Ang^3
  Stress (GPa):xx,   yy,    zz,    yz,    xz,    xy
            -0.004  0.002  0.002-0.000 -0.000 -0.000
 Atom#  sym       position [x,y,z]tag  rmsForce constraints
   0    C   [0.000      0.000      0.000]  0   5.78      T T T
   1    O   [1.100      0.000      0.000]  0   5.78      T T T
--------------------------------------------------

INCAR Parameters:
-----------------
        nbands: 8
        ismear: 1
         encut: 350.0
         sigma: 0.01
        magmom: None
          kpts: [1, 1, 1]
    reciprocal: False
            xc: PBE
           txt: -
         gamma: False

Pseudopotentials used:
----------------------
C: potpaw_PBE/C/POTCAR (git-hash: ee4d8576584f8e9f32e90853a0cbf9d4a9297330)
O: potpaw_PBE/O/POTCAR (git-hash: 592f34096943a6f30db8749d13efca516d75ec55)
```

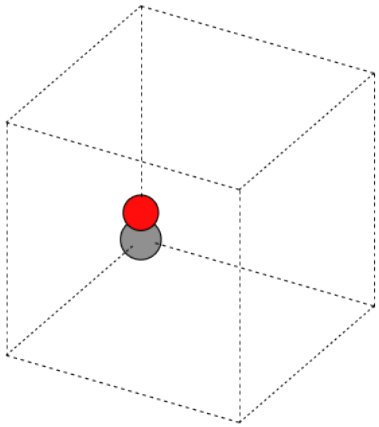
We can also look at the files created by `VASP` to see if everything went ok.

6 Geometry Optimizations

Now let us try to do a geometry optimization. For this `VASP` needs two additional keywords (at least) - `IBRION` and `NSW`. `IBRION` controls the relaxation algorithm and `NSW` specifies the total number of steps.

```
from ase import Atoms, Atom
from ase.io import write
from ase.visualize import view
from jasp import jasp

# define an Atoms object
co = Atoms([Atom('C', [0., 0., 0.]),
                Atom('O', [1.1, 0., 0.])],
               cell=(10, 10, 10))

with jasp('molecules/geometry-co',  # output dir relative to current dir
          xc='PBE',  # the exchange-correlation functional
          nbands=8,    # number of bands
          encut=350,    # planewave cutoff
          ismear=1,    # Methfessel-Paxton smearing
          sigma=0.01,  # very small smearing factor for a molecule
          nsw=20, # Number of ionic steps
          ibrion=2, # Conjugate gradient alogrithm
          atoms=co) as calc:
    print('energy = {0} eV'.format(co.get_potential_energy()))
    print 'Forces (eV/Ang.):'
    print(co.get_forces())
    print 'Equilibrium Positions (Angs.):'
    for atom in co:
        print atom.symbol, atom.position

# Save an image. Note that this is done outside the with statement
write('images/CO-relaxed.png', co, show_unit_cell=2, rotation='60x,-30y,90z')
```

```
energy = -14.81175954 eV
Forces (eV/Ang.):
[[ 0.003  0.     0.   ]
 [-0.003  0.     0.   ]]
Equilibrium Positions (Angs.):
C [-0.022  0.     0.   ]
O [ 1.122  0.     0.   ]
```

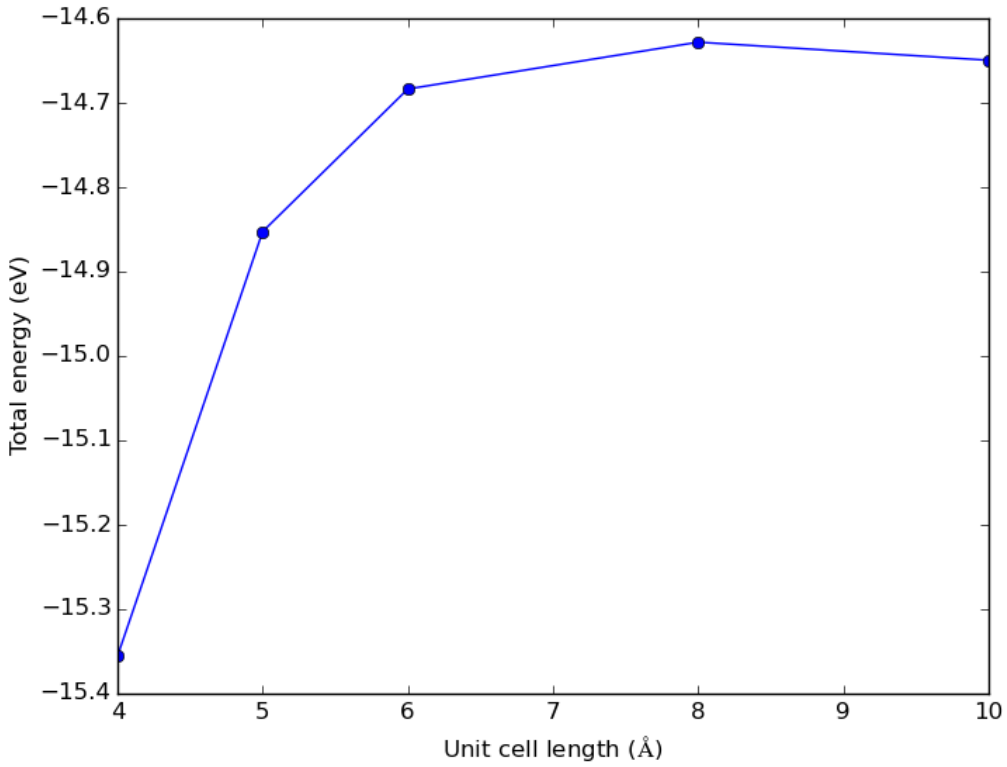


We might also want to visualize the relaxation trajectory. Using the terminal, change into the directory where you performed the calculation, and type in `jaspsum -t`.

7 Effect of Unit Cell Size

Let us consider a more complicated example. Here we will vary the size of the unit cell, to see how interactions between periodic images affect the energy.

```
from jasp import *
from ase import Atoms,Atom
import numpy as np

atoms = Atoms([Atom('C',[0,   0, 0]),
               Atom('O',[1.2, 0, 0])])

L = [4, 5, 6, 8, 10]

energies = []

ready = True

for a in L:
    atoms.set_cell([a,a,a], scale_atoms=False)
    atoms.center()
    with jasp('molecules/co-L-{0}'.format(a),
              encut=350,
              xc='PBE',
              atoms=atoms) as calc:
        try:
            energies.append(atoms.get_potential_energy())
        except (VaspSubmitted, VaspQueued):
            ready = False

if not ready:
    import sys; sys.exit()

import matplotlib.pyplot as plt
plt.plot(L, energies, 'bo-')
plt.xlabel('Unit cell length ($\AA$)')
plt.ylabel('Total energy (eV)')
plt.savefig('images/co-e-v.png')
plt.show()
```



We can see that at small box sizes, there are attractive interactions between CO molecules that lower the total energy. At larger box sizes the energy starts to converge to a fixed value as the interactions are minimized. Now let's check the effect on the computational cost.

```
from jasp import *

L = [4, 5, 6, 8, 10]

for a in L:
    with jasp('molecules/co-L-{0}'.format(a)) as calc:
        print '{0} {1} seconds'.format(a, calc.get_elapsed_time())
```

```
4 2.616 seconds
5 3.907 seconds
6 5.891 seconds
8 16.588 seconds
10 30.543 seconds
```

We can see the computational cost went up by a factor of 15! Perhaps you can now appreciate the computational cost involved in simulating 100s of atoms in large boxes!

8 Miscellaneous

8.1 Building pdfs from org files

Using the software you loaded at the beginning of lab, you should be able to build a pdf from your .org files. Let us try that, click on the Org menu and click Export/Publish. Then press 'l' and 'o'. This let's you build a pdf and open it.

Alternately, you can type, `C-c C-e l o`

8.2 Viewing latex equations in org documents

Click on `org-toggle-latex-overlays`. You should be able to see the Schrodinger equation below.

- $H\psi = E\psi$