Mini Project 1

2012-11-02 Fri

Contents

1 Problem Statement **2**

2 Implementation Code **2**

2.1 Geometry Optimization 2
2.2 Plotting Atoms 3
2.3 Plotting Bonds 3
2.4 Plotting Cloud/Fog 5

3 Instructions and Files **5**

4 Results **6**

4.1 CO2 6
4.2 CO 6
4.3 N2 7
4.4 H2O 7
4.5 CH4 8
4.6 C2H6 8
4.7 C2H4 9
4.8 C2H2 9
4.9 CH3CH2OH 10
4.10 CH3CONH2 10
4.11 C6H6 11
4.12 Ta 11
4.13 TaC 12
4.14 Graphite 12

1 Problem Statement

The problem deals with constructing a fog diagram of the electron/charge density of molecules and atoms in primitive cells. The charge density distribution is obtained through VASP using the 'jasp' code.

2 Implementation Code

2.1 Geometry Optimization

The molecules or primitive cells are relaxed using the following general code set:

```python
from ase import Atom, Atoms
from ase.data.molecules import molecule
from jasp import *

# Create the initial structure
mol1 =  molecule('CH3CONH2')
mol1.set_cell([10, 10, 10], scale_atoms=False)
mol1.center()

ready = True

emol=0

# Make a geometry optimized structure of a molecule and get the charge density
# Default calculations made with molecule at center of unit cell for better visualization
with jasp('molecules/wgs/CH3CONH2-center',
          xc='PBE',
          encut=350,#Energy cutoff
          ismear=0, #Smearing of orbitals
          ibrion=2, #Geometry optimization
          nsw=8,
          nbands=2*6 + 1*3 + 1*3 + 5*1 + 3,
          ispin=2,  #Spind polarization
          atoms=mol1) as calc:
    try:
        emol = mol1.get_potential_energy()
        fmol = mol1.get_forces()
        print calc
    except (VaspSubmitted, VaspQueued):
        ready = False

print emol
print fmol
```

In this code snippet, ibrion=2 is used for geometry optimization. One could also use an equation of state method. The main outcome of this step is that an optimized structure must be obtained for which the charge density distribution file ‘CHGCAR’ has been created.

2.2 Plotting Atoms

The plotting procedure is executed next. First, atoms are plotted as solid spheres:

```python
from jasp import *
from enthought.mayavi import mlab
from ase.data import vdw_radii
from ase.data.colors import cpk_colors
import numpy as np


mol_name = 'CH3CONH2'

#Read molecule data from exisiting folder
with jasp('molecules/wgs/{0}-center'.format(mol_name)) as calc:
    atoms = calc.get_atoms()
    atoms.center()
    x, y, z, cd = calc.get_charge_density()

mlab.figure(bgcolor=(1, 1, 1), size=(350,350)) # make a white figure

atoms_x = []
atoms_y = []
atoms_z = []

# Plot the atoms as spheres
for atom in atoms:
    mlab.points3d(atom.x,
                  atom.y,
                 atom.z,
                   scale_factor=vdw_radii[atom.number]/2., #this determines the size of the atom
                   resolution=20,
                   # a tuple is required for the color
                   color=tuple(cpk_colors[atom.number]),
                   scale_mode='none')
    atoms_x.append(atom.x)
    atoms_y.append(atom.y)
    atoms_z.append(atom.z)
    #print vdw_radii[atom.number]/2.
```

2.3 Plotting Bonds

Plotting the bonds in the case of molecules is slightly complicated. To figure out the number and participants in the bonds, the nearest neighbours of each atom must be known. This can either be calculated explicitly or can be read from the ‘OUTCAR’ file. Since reading the file is probably computationally more efficient, I have used that method to create the nearest neighbour list. From this list, covalent bonds for each atom are drawn to its nearest neighbours. The code for this functionality is as follows:

```python
#Plot bonds as tubes
#Get positions of atom
bonds_pos = []
f = open('molecules/wgs/{0}-center/OUTCAR'.format(mol_name),'r')
while ('ion  position' not in f.readline()):
    pass
r1 = np.arange(len(atoms_x))
for i in r1:
    a = f.readline()
    data1 = np.fromstring(a,sep='  ')
    bonds_pos = np.append(bonds_pos,data1[1:])
    i+=1
f.close()
bonds_pos *=10

#Get nearest neighbour list to draw bonds
f = open('molecules/wgs/{0}-center/OUTCAR'.format(mol_name),'r')
while ('ion  position' not in f.readline()):
    pass
r1 = np.arange(len(atoms_x))
for i in r1:
    a = f.readline()
    index = 0
    for a1 in a:
        if '-' not in a1:
            index +=1
        else:
            break
    a2 = a[index+1:]
    data2 = np.fromstring(a2, sep=' ')
    r2 = np.arange(0,len(data2),2)
    for j in r2:
        ind = int(data2[j])-1
        mlab.plot3d([bonds_pos[ind*3 + 0],bonds_pos[i*3 + 0]],[bonds_pos[ind*3 + 1],bonds_pos[i*3 + 1]],
                    [bonds_pos[ind*3 + 2],bonds_pos[i*3 + 2]],tube_radius=0.07, colormap='Reds')
    i+=1
f.close()
```

This code does not distinguish between single/double/triple bonds and will only draw a single tube for each case. However, when looking at the electron density, the difference between these various kinds of bonds can be observed (I have plotted cases of C2H6, C2H4 and C2H2) Also, it can be noticed that the positions of the atoms are read again (they were previously obtained using get_atoms() for plotting the atoms). This is because the atom list in 'OUTCAR' is not similar to the atom list output using get_atoms().

2.4 Plotting Cloud/Fog

Lastly, the charge density cloud is plotted using volume rendering via the following code:

```
#Draw electron density as fog
source = mlab.pipeline.scalar_field(x,y,z,cd)
min = cd.min()
max = cd.max()
vol = mlab.pipeline.volume(source, vmin=min+0.008*(max-min), vmax=min + 0.1*(max-min))

#Save image at different angles
mlab.view(azimuth=0, elevation=90, distance=10)
mlab.savefig('images/{0}_1.png'.format(mol_name.lower()))
mlab.view(azimuth=90, elevation=0, distance=10)
mlab.savefig('images/{0}_3.png'.format(mol_name.lower()))
mlab.show()
```

The 'vmax' and 'vmin' parameters are set to adjust how the cloud is visualized. They had to be adjusted a significant amount and the present combination seemed to provide the best results in terms of how well the charge density features could be viewed.

3 Instructions and Files

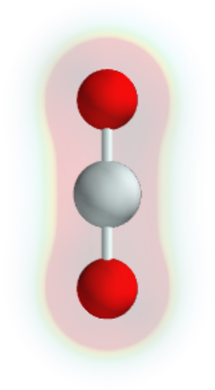
The geometry optimization code is present in the file 'mol.org'. The 'mol_name' variable needs to be changed accordingly by the user. Also, the parameters for optimization should be adjusted as required. Upon execution, the code will create a folder 'mol_name-center' in the 'molecules' folder.

To create the images, there are two files present - 'viz_mol.org' and 'vis_latt.org' for molecules and crystal structures respectively. They will only work if the optimized folders created by 'mol.org' or from any other source is present in the 'molecules' folder. In each of these files, the user must specify the molecule/crystal whose charge density he wants to visualize using the 'mol_name' variable. Upon execution, image files 'mol_name_1.png' and 'mol_name_2.png' will be created in the 'images' folder. These images have different azimuth and elevation parameters. The user can adjust these as required if he wants to.

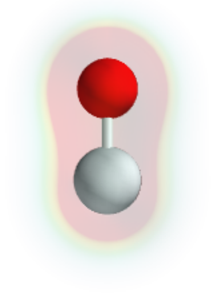
4 Results

The following are images of the molecules studied, the asymmetrical ones having two viewing directions.

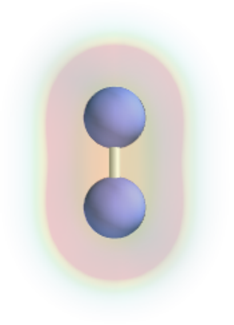
4.1 CO2



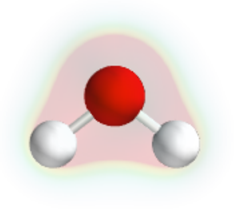
4.2 CO



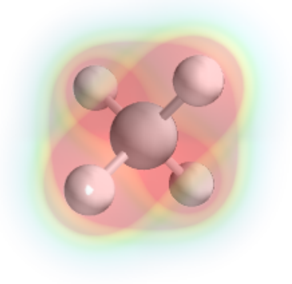
4.3 N2



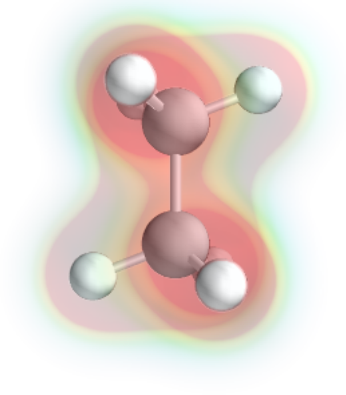
4.4 H2O

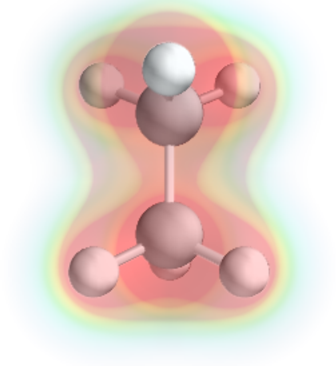


4.5 CH4

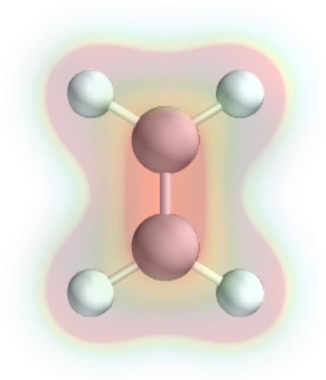
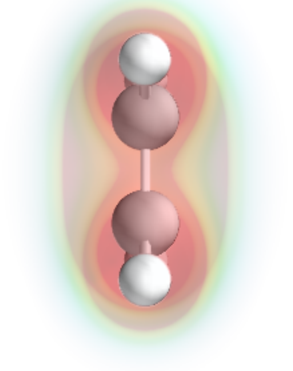


4.6 C2H6

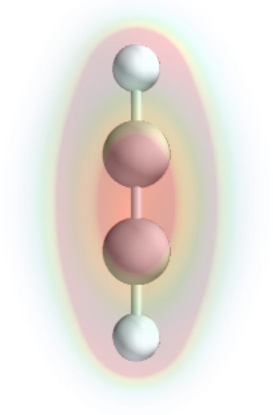
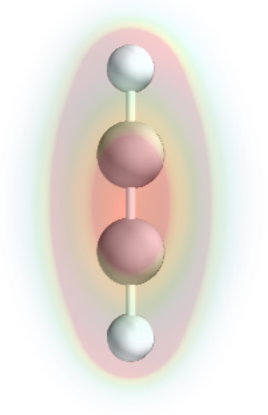




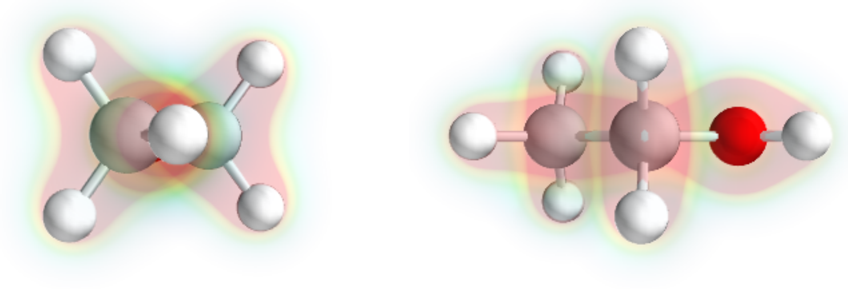
4.7 C2H4

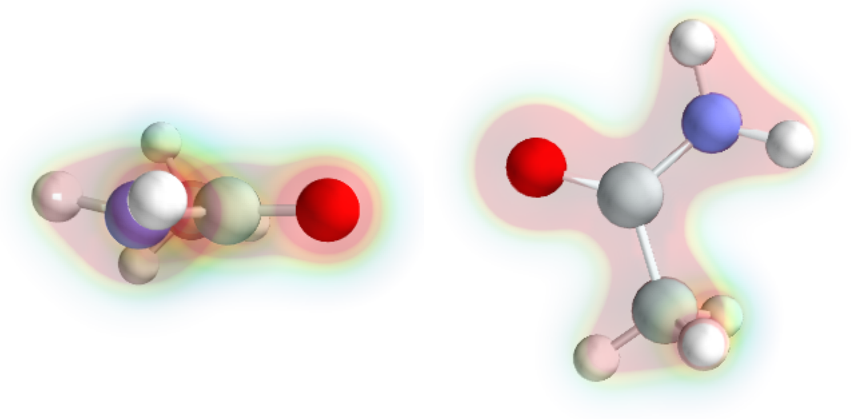
4.8 C2H2

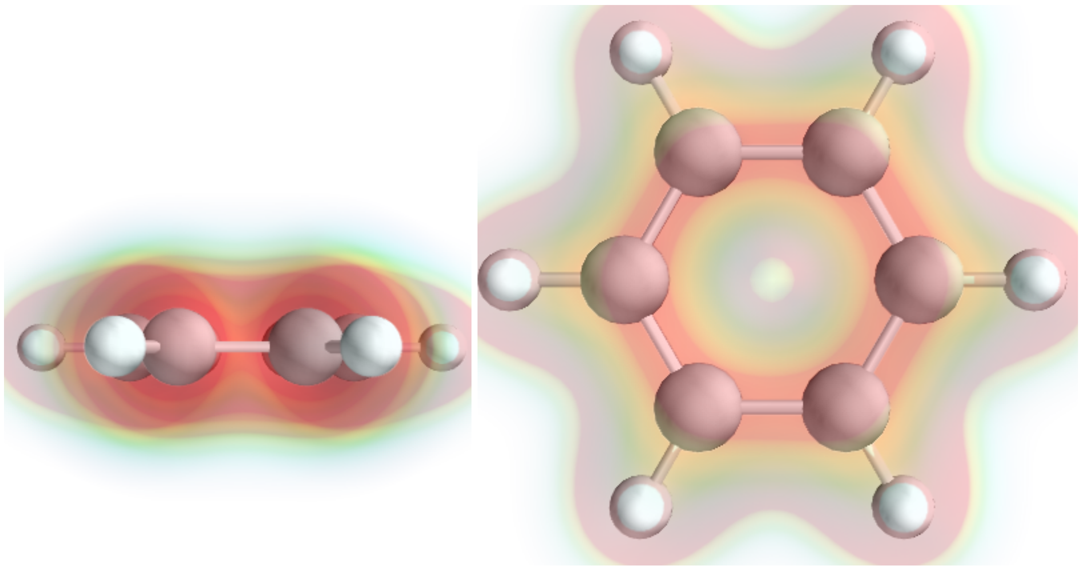
4.9 CH3CH2OH



4.10 CH3CONH2

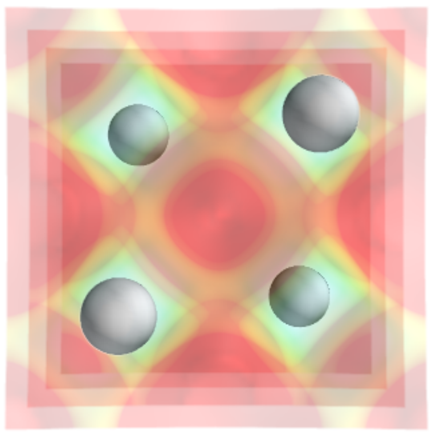


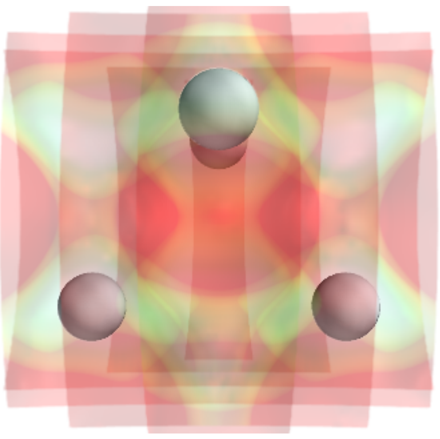
4.11 C6H6



4.12 Ta

4.13 TaC





4.14 Graphite