

Computational Chemistry Laboratory IV (CBE 60547)

Prateek Mehta, William F. Schneider

<2015-03-24 Tue>

1 Common sources of error from the last homework!

Mostly everyone did a good job on the last homework. There were a few common mistakes, listed below.

- Not checking if results make sense
 - e.g. if the calculation took only one relaxation step, the geometry is probably not converged
 - always a good idea to check with your input files and output files
- Bond energies
 - e.g. for oxygen, bond energy is $E_{O_2} + E_{ZPE} - 2E_O$
 - absolute DFT energies are meaningless!
- Use consistent energy cutoffs when comparing different calculations.
- Typos, spelling mistakes, forgetting to change directory names
- **jasp** is built for continuation runs
 - **jasp** reads the input parameters from the files in the directory
 - jobs are resubmitted if parameters are added or updated
 - If parameters are removed, the original parameters written in your directory **will stay!**
 - Changing atoms objects after running the code is a bad idea! (**jasp** got updated to raise errors if you do this)
- Those not submitting org generated pdfs, make sure you include enough information (incar parameters, pseudopotentials used, etc.) to make your calculations reproducible

2 Bulk Systems

2.1 Creating a bulk system

There are a few different functions built into ase that let's you create simple bulk systems. The one we will use is [ase.lattice.bulk](#). Let us consider how to construct an fcc structure. You can either create the primitive cell (default), a orthorhombic cell, or a cubic cell. Note these will have different unit cell sizes and different number of atoms, which will affect the computational cost of your simulation.

```
1 from ase.lattice import bulk
2 from jasp import *
3 import matplotlib.pyplot as plt
4 from ase.visualize import view
5
6 # a is the lattice constant in Angstroms
7 Pd_cubic = bulk('Pd', 'fcc', a=3.89, cubic=True)
8 view(Pd_cubic)
9 print Pd_cubic
10
11 Pd_ortho = bulk('Pd', 'fcc', a=3.89, orthorhombic=True)
12 view(Pd_ortho)
13 print Pd_ortho
14
15 Pd_primitive = bulk('Pd', 'fcc') # if a is not specified, default experimental value is used.
16 view(Pd_primitive)
17 print Pd_primitive
```

```
Atoms(symbols='Pd4', positions=..., cell=[3.89, 3.89, 3.89],
      pbc=[True, True, True])
Atoms(symbols='Pd2', positions=..., cell=[2.75064537881567,
      2.75064537881567, 3.89], pbc=[True, True, True])
Atoms(symbols='Pd', positions=..., cell=[[0.0, 1.945, 1.945], [1.945,
      0.0, 1.945], [1.945, 1.945, 0.0]], pbc=[True, True, True])
```

Calculations can be done in a similar way to what we performed in the last lab and homework, with a few additional parameters. The most important of these is the k -point grid.

2.2 A little bit about k -points

For modeling bulk systems/surfaces, we need to specify a k -point mesh. A few things to note:

- The accuracy (and cost) of your simulation usually goes up with increasing the k -point grid used.
- In this lab (for demonstrative purposes only) a k -point grid of (8x8x8) has been used. We typically need to do a convergence study for k -points, similar to the one we did for the planewave cutoff.
- The number of k -points required scales inversely with the size of the unit cell.
- We will usually specify a parameter `kpts=(k1,k2,k3)` in `jasp`, which creates a KPOINTS file that tells VASP to [automatically create a Monkhorst-Pack \$k\$ -mesh](#).

- Metals (conductors) generally require more k -points than insulators to reach the same level of convergence.

2.3 Optimizing lattice constants - Equations of State

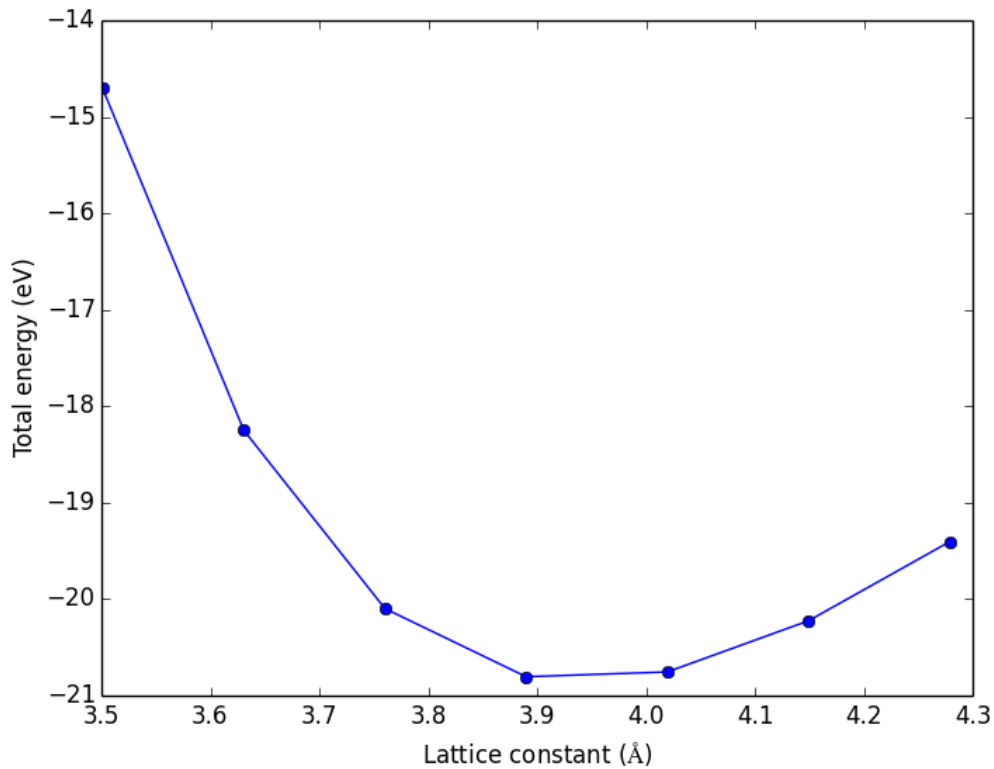
2.3.1 Energies vs. Lattice Constants

`ase` by default uses the experimental lattice constants (if known). Our computationally calculated lattice constants will usually be slightly different, depending on the exchange-correlation functional used. Let us consider a series of lattice constants, to find out which one minimizes the total energy.

```

1  from ase.lattice import bulk
2  from jasp import *
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from ase.utils.eos import EquationOfState
6
7  # Generate an array of 7 points 10% around 3.89 angstroms
8  A = np.linspace(3.89 * 0.9, 3.89 * 1.1, 7)
9
10 energies = []
11
12 ready = True
13 for a in A:
14     # We will use the cubic cell for simplicity
15     Pd_cubic = bulk('Pd', 'fcc', a=a, cubic=True)
16
17     with jasp('EOS/Pd-a-{:0:1.2f}'.format(a),
18              xc='PBE',
19              encut=400,
20              ismear=1, # Use MP smearing for metals
21              kpts=(8,8,8), #A much larger grid is reqd to be accurate!
22              atoms=Pd_cubic) as calc:
23         try:
24             calc.calculate()
25             energies.append(Pd_cubic.get_potential_energy())
26
27         except(VaspSubmitted, VaspQueued):
28             ready = False
29
30 if not ready:
31     import sys; sys.exit()
32
33 plt.plot(A, energies, 'bo-')
34 plt.xlabel('Lattice constant ($AA$)')
35 plt.ylabel('Total energy (eV)')
36 plt.savefig('images/Pd-fcc-lattice.png')
37 plt.show()

```



2.3.2 Fitting to an Equation of State

To find the 'optimal' lattice constant we need to fit our data to an [equation of state](#), which describes the energy as a function of volume. The Murnaghan or Birch-Murnaghan EOS is commonly used. Let us use [ase.utils.eos](#) to fit the data we calculated above to the Birch-Murnaghan EOS.

```

1  from jasp import *
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from ase.utils.eos import EquationOfState
5
6  # Generate an array of 7 points 10% around 3.89
7  A = np.linspace(3.89 * 0.9, 3.89 * 1.1, 7)
8
9  energies = []
10 volumes = []
11
12 for a in A:
13
14     with jasp('EOS/Pd-a-{0:1.2f}'.format(a)) as calc:
15         atoms = calc.get_atoms()
16         energies.append(atoms.get_potential_energy())
17         volumes.append(atoms.get_volume())
18
19 eos = EquationOfState(volumes, energies, eos='birchmurnaghan')
20 v0, e0, b = eos.fit()
21 a0 = v0 ** (1/3.)
22 eos.plot(filename='images/Pd-EOS.png', show=True)
23
24 print 'Minimum Energy = {0:1.3f} eV'.format(e0)

```

```

25 print 'Optimal Volume = {0:1.3f} cubic angstroms'.format(v0)
26 print 'Optimal lattice constant = {0:1.3f} angstroms'.format(a0)

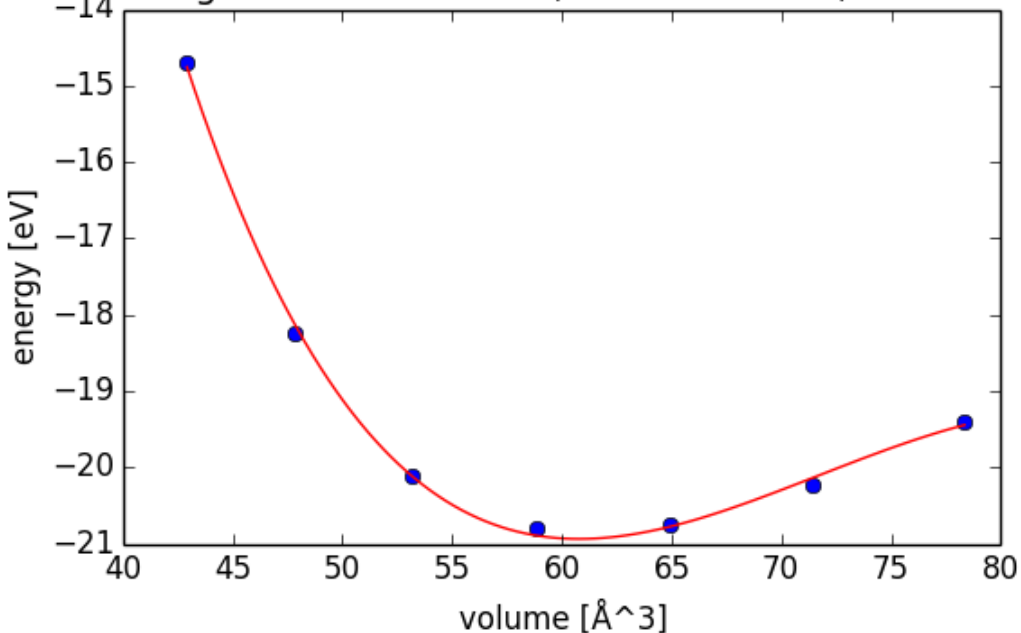
```

```

Minimum Energy = -20.933 eV
Optimal Volume = 60.782 cubic angstroms
Optimal lattice constant = 3.932 angstroms

```

birchmurnaghan: E: -20.933 eV, V: 60.782 Å³, B: 213.276 GF



3 Surfaces

3.1 Creating a surface

`ase` provides functions to create surfaces too. Surfaces are layers of atoms formed by cleaving the bulk structure in a given direction. In our models, we add vacuum in the direction perpendicular to the surface. Thus, the atoms are finite in the direction perpendicular to the surface, but infinite in the other two directions. Here is an example of how to make a surface.

```

1 from ase.lattice.surface import fcc111
2 from jasp import *
3 from ase.visualize import view
4 from ase.constraints import FixAtoms
5
6 a = 3.932 # Optimal lattice constant from EOS
7
8 # Create a surface with 3 unit cells in x and y
9 # 3 layers deep
10 atoms = fcc111('Pd', size=(2,2,3), vacuum=10.0, a=a)
11 view(atoms)
12 for atom in atoms:
13     print atom
14 write('images/Pd-slab.png', atoms, rotation='90x', show_unit_cell=2)

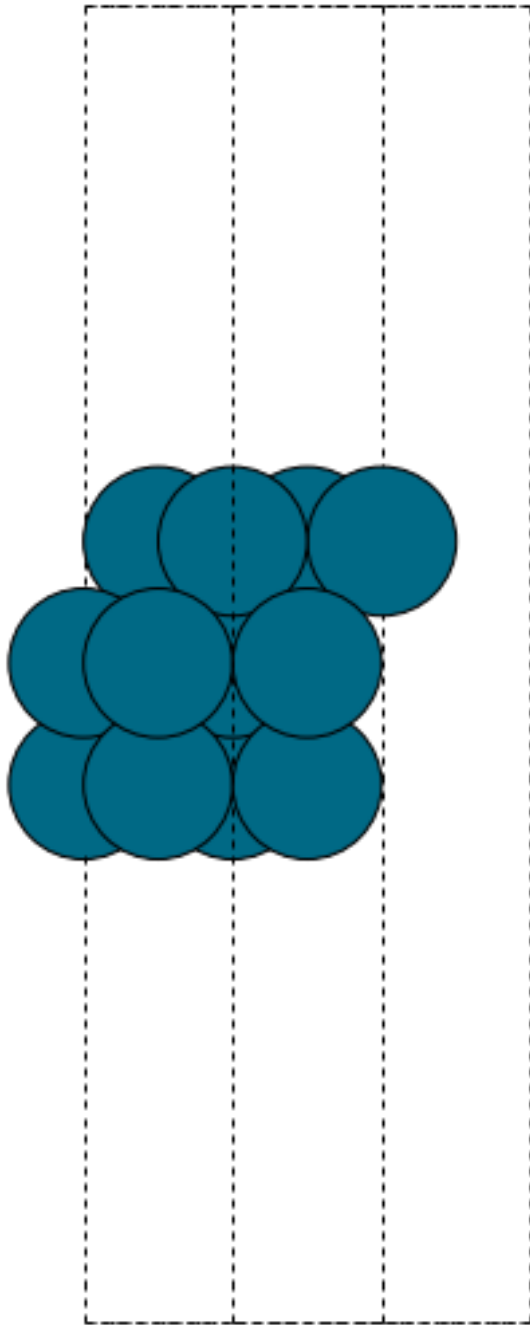
```

```

Atom('Pd', [1.3901719318127526, 0.8026161390519545, 10.0], tag=3, index=0)
Atom('Pd', [4.1705157954382575, 0.8026161390519545, 10.0], tag=3, index=1)
Atom('Pd', [2.7803438636255051, 3.210464556207818, 10.0], tag=3, index=2)
Atom('Pd', [5.5606877272510102, 3.210464556207818, 10.0], tag=3, index=3)
Atom('Pd', [0.0, 1.605232278103909, 12.270141258453609], tag=2, index=4)
Atom('Pd', [2.7803438636255051, 1.605232278103909, 12.270141258453609], tag=2, index=5)
Atom('Pd', [1.3901719318127523, 4.013080695259772, 12.270141258453609], tag=2, index=6)
Atom('Pd', [4.1705157954382575, 4.013080695259772, 12.270141258453609], tag=2, index=7)
Atom('Pd', [0.0, 0.0, 14.540282516907219], tag=1, index=8)
Atom('Pd', [2.7803438636255051, 0.0, 14.540282516907219], tag=1, index=9)
Atom('Pd', [1.3901719318127526, 2.4078484171558636, 14.540282516907219], tag=1, index=10)
Atom('Pd', [4.1705157954382575, 2.4078484171558636, 14.540282516907219], tag=1, index=11)

```

The tag on the atom indicates which layer of the surface it is in.



We can see that there are actually two surfaces, one in the top layer and one at the bottom layer. Surface atoms will tend to contract toward the bulk due to decreased coordination.

To simulate bulk like behavior in regions away from the surface, we can do two things:

- increase the number of layers in the slab (requires many atoms, large cost)
- Constrain(freeze) the the bottom layer(s) in their bulk positions (common, lower cost). The bottom layer is now representative of bulk behavior.

3.2 Surface calculations

Let us now optimize the geometry of our surface. Note that only one k -point is required in the direction perpendicular to the surface.

```
1 from ase.lattice.surface import fcc111
2 from jasp import *
3 from ase.visualize import view
4 from ase.constraints import FixAtoms
5
6 JASPRC['queue.nprocs'] = 8
7 JASPRC['queue.q'] = 'short'
8
9 a = 3.932 # Optimal lattice constant from EOS
10 atoms = fcc111('Pd', size=(2,2,3), vacuum=10.0, a=a)
11
12 constraint = FixAtoms(mask=[atom.tag >= 3 for atom in atoms])
13 atoms.set_constraint(constraint)
14
15 with jasp('surfaces/Pd-slab-relaxed',
16          xc='PBE',
17          ismear=1,
18          kpts=(8, 8, 1),
19          encut=400,
20          ibrion=2, # Conjugate Gradient
21          nsw=20, # relaxation steps
22          atoms=atoms) as calc:
23     calc.calculate()
24     print calc
```

: -----

VASP calculation from /afs/crc.nd.edu/user/p/pmehta1/computational-chemistry/Lab4/surfaces/Pd

converged: True

Energy = -58.019294 eV

Unit cell vectors (angstroms)

	x	y	z	length
--	---	---	---	--------

a0	[5.561	0.000	0.000]	5.561
----	---------	-------	--------	-------

a1	[2.780	4.816	0.000]	5.561
----	---------	-------	--------	-------

a2	[0.000	0.000	24.540]	24.540
----	---------	-------	---------	--------

a,b,c,alpha,beta,gamma (deg):5.561 5.561 24.540 90.0 90.0 90.0

Unit cell volume = 657.154 Ang³

Stress (GPa):xx, yy, zz, yz, xz, xy
0.012 0.012 0.000-0.000 -0.000 -0.000

Atom#	sym	position [x,y,z]	tag	rmsForce	constraints
-------	-----	------------------	-----	----------	-------------

0	Pd	[1.390 0.803 10.000]	3	0.00	F F F
---	----	----------------------	---	------	-------

1	Pd	[4.171 0.803 10.000]	3	0.00	F F F
---	----	----------------------	---	------	-------

2	Pd	[2.780 3.210 10.000]	3	0.00	F F F
---	----	----------------------	---	------	-------

3	Pd	[5.561 3.210 10.000]	3	0.00	F F F
---	----	----------------------	---	------	-------

4	Pd	[5.561 1.605 12.270]	2	0.04	T T T
---	----	----------------------	---	------	-------

5	Pd	[2.780 1.605 12.270]	2	0.04	T T T
---	----	----------------------	---	------	-------

6	Pd	[6.951 4.013 12.270]	2	0.04	T T T
---	----	----------------------	---	------	-------

7	Pd	[4.171 4.013 12.270]	2	0.04	T T T
---	----	----------------------	---	------	-------

8	Pd	[0.000 0.000 14.548]	1	0.02	T T T
---	----	----------------------	---	------	-------

9	Pd	[2.780 0.000 14.548]	1	0.02	T T T
---	----	----------------------	---	------	-------

10	Pd	[1.390	2.408	14.548]	1	0.02	T T T
11	Pd	[4.171	2.408	14.548]	1	0.02	T T T

INCAR Parameters:

```

nbands: 72
ismear: 1
  nsw: 20
ibrion: 2
  encut: 400.0
magnom: None
  kpts: (8, 8, 1)
reciprocal: False
  xc: PBE
  txt: -
gamma: False

```

Pseudopotentials used:

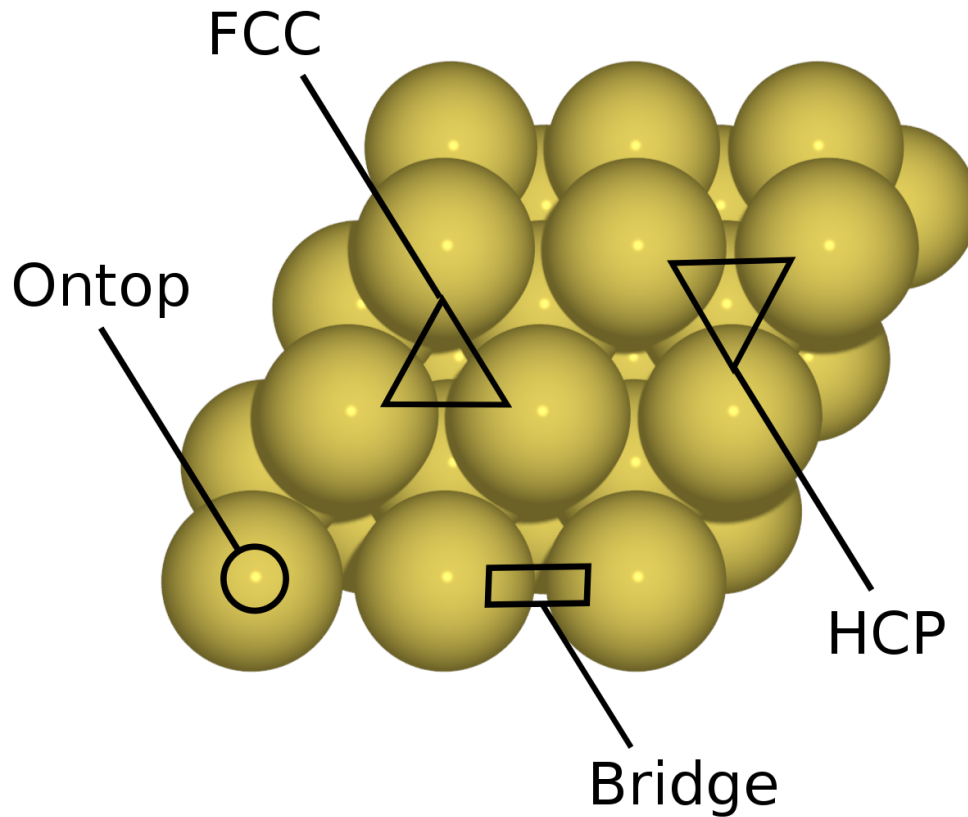
Pd: potpaw_PBE/Pd/POTCAR (git-hash: 04426435b178dfad58ed91b470847d50ff70b858)

Note that Vasp is a little unintuitive. The constraint 'F' means frozen.

We can go back to the calculation directory and see how our surface relaxed with `jaspsum -t`.

3.3 Adding an Adsorbate

Now let us add an adsorbate on our surface. There are multiple places where it could adsorb. Here is a picture of a fcc(111) gold surface, showing the possible adsorption sites.



Let's go back to our Pd surface and perform a calculation with an Oxygen adsorbate at the fcc site.

```

1  from ase.lattice.surface import fcc111, add_adsorbate
2  from jasp import *
3  from ase.visualize import view
4  from ase.constraints import FixAtoms
5
6  a = 3.932 # Optimal lattice constant from EOS
7  atoms = fcc111('Pd', size=(2,2,3), vacuum=10.0)
8
9  add_adsorbate(atoms, 'O', height=1.2, position='fcc')
10
11 # Note that constraints only work after adding the adsorbate
12 constraint = FixAtoms(mask=[atom.tag >= 3 for atom in atoms])
13 atoms.set_constraint(constraint)
14
15 # view(atoms)
16 with jasp('surfaces/O-on-Pd-fcc',
17         xc='PBE',
18         ismear=1,
19         kpts=(8, 8, 1),
20         encut=400,
21         ibrion=2, # Conjugate Gradient
22         nsw=20, # relaxation steps

```

```

23         atoms=atoms) as calc:
24     calc.calculate()
25     print calc
26
27 write('images/Pd-slab-0-fcc.png', atoms, show_unit_cell=2)

```

```

: -----
VASP calculation from /afs/crc.nd.edu/user/p/pmehta1/computational-chemistry/Lab4/surfaces/0
converged: True
Energy = -64.436725 eV

```

Unit cell vectors (angstroms)

	x	y	z	length
a0	[5.501	0.000	0.000]	5.501
a1	[2.751	4.764	0.000]	5.501
a2	[0.000	0.000	24.492]	24.492

a,b,c,alpha,beta,gamma (deg):5.501 5.501 24.492 90.0 90.0 90.0

Unit cell volume = 641.919 Ang³

Stress (GPa):xx, yy, zz, yz, xz, xy
0.004 0.004 0.002-0.000 -0.000 -0.000

Atom#	sym	position [x,y,z]	tag	rmsForce	constraints
0	Pd	[1.375 0.794 10.000]	3	0.00	F F F
1	Pd	[4.126 0.794 10.000]	3	0.00	F F F
2	Pd	[2.751 3.176 10.000]	3	0.00	F F F
3	Pd	[5.501 3.176 10.000]	3	0.00	F F F
4	Pd	[5.505 1.586 12.288]	2	0.02	T T T
5	Pd	[2.747 1.586 12.288]	2	0.02	T T T
6	Pd	[6.877 3.970 12.405]	2	0.03	T T T
7	Pd	[4.126 3.975 12.288]	2	0.02	T T T
8	Pd	[-0.031 -0.018 14.660]	1	0.01	T T T
9	Pd	[2.782 -0.018 14.660]	1	0.01	T T T
10	Pd	[1.375 2.418 14.660]	1	0.01	T T T
11	Pd	[4.126 2.382 14.541]	1	0.01	T T T
12	O	[1.375 0.794 15.820]	0	0.02	T T T

INCAR Parameters:

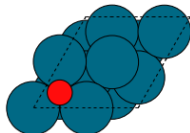
```

nbands: 80
ismear: 1
nsw: 20
ibrion: 2
encut: 400.0
magmom: None
kpts: (8, 8, 1)
reciprocal: False
xc: PBE
txt: -
gamma: False

```

Pseudopotentials used:

O: potpaw_PBE/O/POTCAR (git-hash: 592f34096943a6f30db8749d13efca516d75ec55)
Pd: potpaw_PBE/Pd/POTCAR (git-hash: 04426435b178dfad58ed91b470847d50ff70b858)



3.4 Calculating adsorption energies

The adsorption energy is given by $E_{ads} = E_{surface+O} - E_{surface} - 0.5E_{O_2}$. This can easily be calculated from the two calculations we performed and the O_2 calculation from the last homework.

4 Density of States

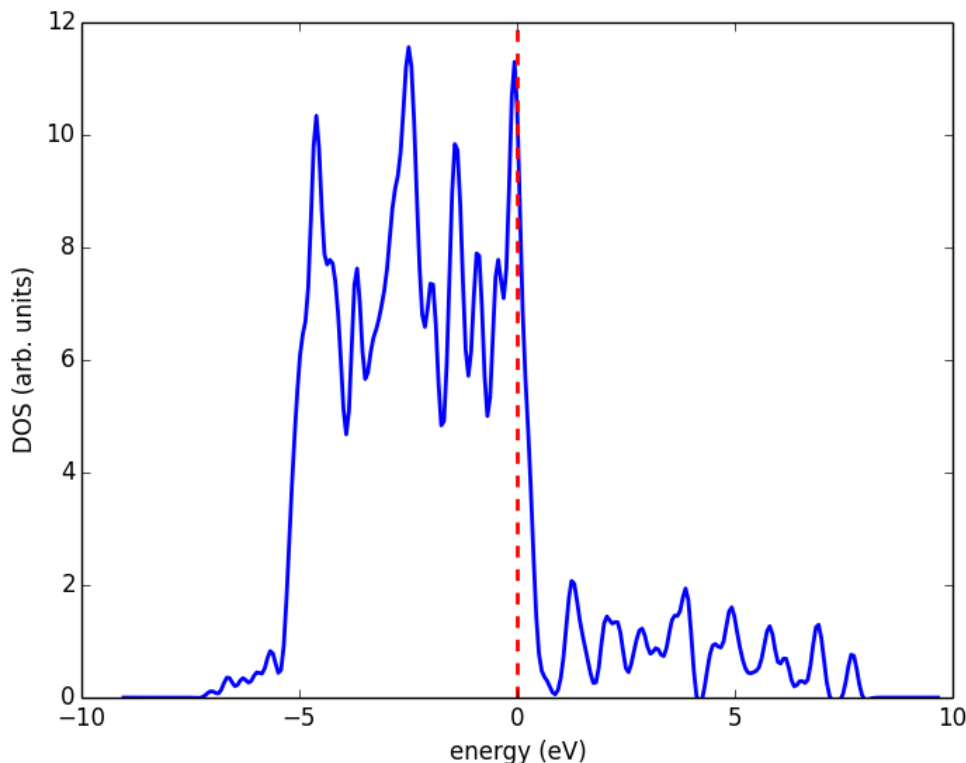
It is possible to plot out the density of states (DOS) from VASP calculations. The density of states describes the number of states per interval of energy at each energy level that are available to be occupied (Wikipedia).

4.1 Total density of States

We can get the total density of states from an old DFT calculation without having to run a new calculation (Though the VASP manual recommends an additional run at `ismear=-5`). The DOS depends on the k -point grid you choose.

Let's read in our calculation from our bulk Pd lattice constant studies.

```
1 from jasp import *
2 from ase.calculators.vasp import VaspDos
3 import matplotlib.pyplot as plt
4
5 with jasp('EOS/Pd-a-3.89') as calc:
6     # Get the dos referenced at the fermi level
7     dos = VaspDos(efermi=calc.get_fermi_level())
8
9 energies = dos.energy
10 dos = dos.dos
11
12 plt.plot(energies, dos, linewidth=2)
13 # Add a vertical line at the fermi level
14 plt.axvline(x=0, color='r', linestyle='--', linewidth=2)
15 plt.ylim(0,12)
16 plt.xlabel('energy (eV)')
17 plt.ylabel('DOS (arb. units)')
18 plt.savefig('images/Pd-bulk-dos.png')
19 plt.show()
```



States to the left of the fermi level (indicated by the red line) are the occupied states.

4.2 Atom-projected density of states

To figure out which density of states belong to which atoms in a molecule, we need to perform an additional calculation. We can compute the atom-projected density of states (ADOS), which is done by projecting the wave function onto localized atomic orbitals. These are only a qualitative representation of the orbitals, because the atoms will often form molecular orbitals, hybridize, etc.

In VASP we can specify an [RWIGS](#) parameter, which is radius around the atom at which to cutoff the projection. The choice of RWIGS is somewhat arbitrary, one can choose the ionic radius of an atom, or a value that minimizes overlap of neighboring spheres. Another way to calculate the ADOS is by specifying the [LORBIT](#) parameter to be 10 or 11, but this only works for PAW potentials (this is what we will use).

In transition metals, the s and p states are dispersed, and the only states that matter in terms of bonding are the d-states. Here is an example to plot the DOS projected on to the d states for clean Pd surface atoms.

```

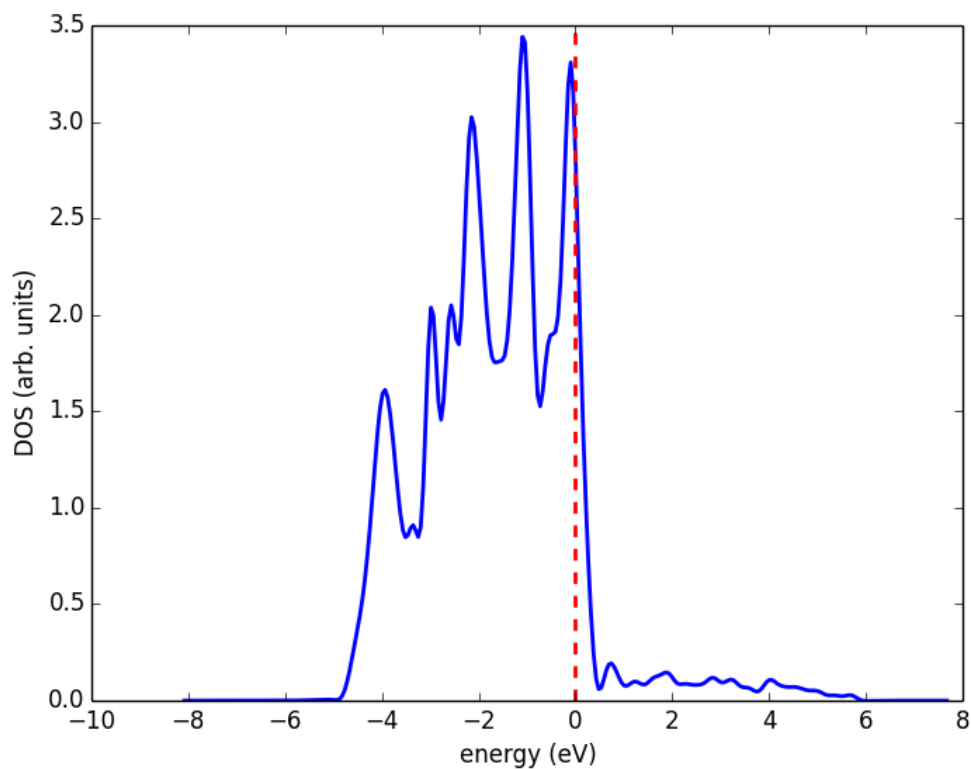
1 from jasp import *
2 from ase.calculators.vasp import VaspDos
3 import matplotlib.pyplot as plt
4
5 # get the geometry the previous calculation
6 with jasp('surfaces/Pd-slab-relaxed') as calc:
```

```

7     atoms = calc.get_atoms()
8
9     #Now submit a calculation for the ados
10    with jasp('surfaces/Pd-ados',
11             xc='PBE',
12             ismear=1,
13             kpts=(8, 8, 1),
14             encut=400,
15             lorbit=10,
16             atoms=atoms) as calc:
17        calc.calculate()
18
19        ados = VaspDos(efermi=calc.get_fermi_level())
20        energies = ados.energy
21        # Atom index 10 is a surface atom (tag=1)
22        print atoms[10]
23        d_dos = ados.site_dos(10, 'd')
24
25        plt.plot(energies, d_dos, lw=2)
26
27        plt.axvline(lw=2, ls='--', color='r')
28        plt.ylim(0, 3.5)
29        plt.xlabel('energy (eV)')
30        plt.ylabel('DOS (arb. units)')
31        plt.savefig('images/Pd-ados.png')
32        plt.show()

```

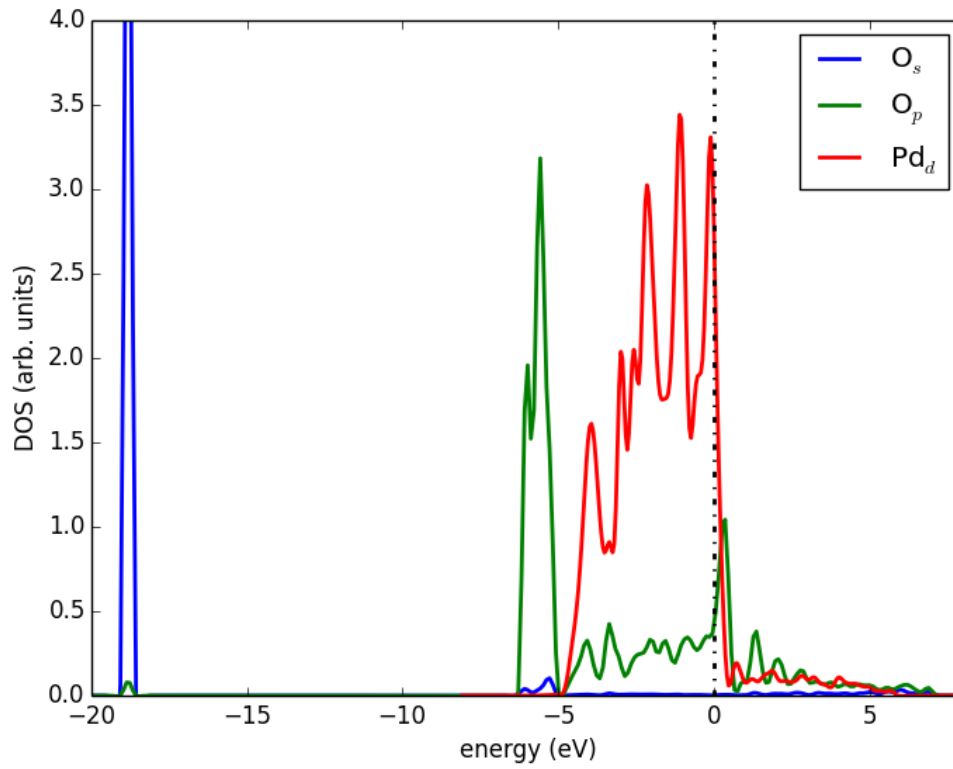
Atom('Pd', [1.3901719318127526, 2.4078484171558636, 14.548402978407839], tag=1, index=10)



4.3 Adsorbate density of states

Now let us plot the density of states for the adsorbed O atom.

```
1  from jasp import *
2  import matplotlib.pyplot as plt
3
4  # get the geometry the previous calculation
5  with jasp('surfaces/O-on-Pd-fcc') as calc:
6      atoms = calc.get_atoms()
7
8  JASPRC['queue.q'] = 'short'
9
10 #Now submit a calculation for the ados
11 with jasp('surfaces/O-on-Pd-fcc-ados',
12          xc='PBE',
13          ismear=1,
14          kpts=(8, 8, 1),
15          encut=400,
16          lorbit=10,
17          atoms=atoms) as calc:
18     calc.calculate()
19
20     O_ados = VaspDos(efermi=calc.get_fermi_level())
21     energies = O_ados.energy
22     # Plot the O ados
23     # 12 is the index of the O atom
24     s_dos = O_ados.site_dos(12, 's')
25     p_dos = O_ados.site_dos(12, 'p')
26     plt.plot(energies, s_dos, label='O$_{s}$', lw=2)
27     plt.plot(energies, p_dos, label='O$_{p}$', lw=2)
28
29 # Now plot the clean surface ados for comparison
30 with jasp('surfaces/Pd-ados') as calc:
31     ados = VaspDos(efermi=calc.get_fermi_level())
32     energies = ados.energy
33
34     d_dos = ados.site_dos(11, 'd')
35     plt.plot(energies, d_dos, label='Pd$_{d}$', lw=2)
36 plt.xlim(-20, 8)
37 plt.ylim(0, 4)
38 plt.axvline(ls='-.', color='k', lw=2)
39 plt.xlabel('energy (eV)')
40 plt.ylabel('DOS (arb. units)')
41 plt.legend()
42 plt.savefig('images/adsorbate-dos.png')
43 plt.show()
```



The blue line indicates the Oxygen s-states. The two peaks of the green line left and right of the Pd d-band are the bonding and antibonding Oxygen p-states. Note that the antibonding peak is to the right of the fermi level, meaning that the antibonding states are unoccupied.