Problem 1

Subproblem 1.1

The code and graph are shown below. The cut-off energy of 300 eV is sufficient for a convergence of 10 meV.

```
from ase.lattice.cubic import BodyCenteredCubic
from jasp import *

atoms = BodyCenteredCubic('Ta')
ENCUTS = [250, 300, 350, 400, 450, 500, 550, 600]
TE = []

ready = True
for e in ENCUTS:
    with jasp('bulk/hw04/Ta-encuts-{0}'.format(e),
              xc='PBE',
              kpts=(10, 10, 10),
              encut=e,
              atoms=atoms) as calc:
        try:
            TE.append(atoms.get_potential_energy())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

import matplotlib.pyplot as plt
TE = np.array(TE)
TE -= TE.min()
plt.plot(ENCUTS, TE, 'o')
plt.xlabel('Cut-off Energy [eV]')
plt.ylabel('Total Energy [eV]')
plt.savefig('images/Ta-encut-convergence.png')
plt.show()
```

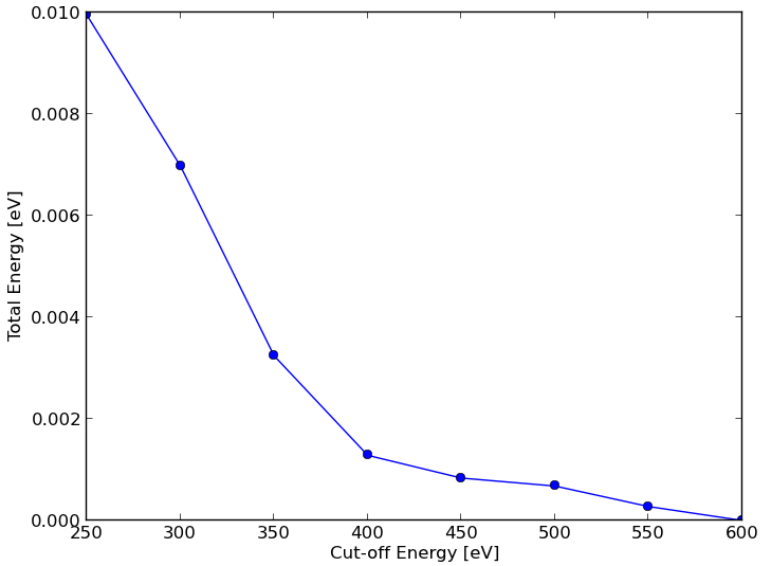

Figure 1: Convergence of energies for Subproblem 1.1.

Subproblem 1.2

The code and graph are shown below. A k-point grid of $6\times6\times6$ is sufficient for a convergence of 50 meV.

```
from ase.lattice.cubic import BodyCenteredCubic
from jasp import *

atoms = BodyCenteredCubic('Ta')
kpts = [2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20]
TE = []

ready = True
for k in kpts:
    with jasp('bulk/hw04/Ta-kpts-{0}'.format(k),
              xc='PBE',
              kpts=(k, k, k),
              encut=300, # From part 1
              atoms=atoms) as calc:
        try:
            TE.append(atoms.get_potential_energy())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

import matplotlib.pyplot as plt
TE = np.array(TE)
TE -= TE.min()
plt.plot(kpts, TE, 'o')
plt.xlabel('Number of k-points in each dimension')
plt.ylabel('Total Energy [eV]')
plt.savefig('images/Ta-kpts-convergence.png')
plt.show()
```

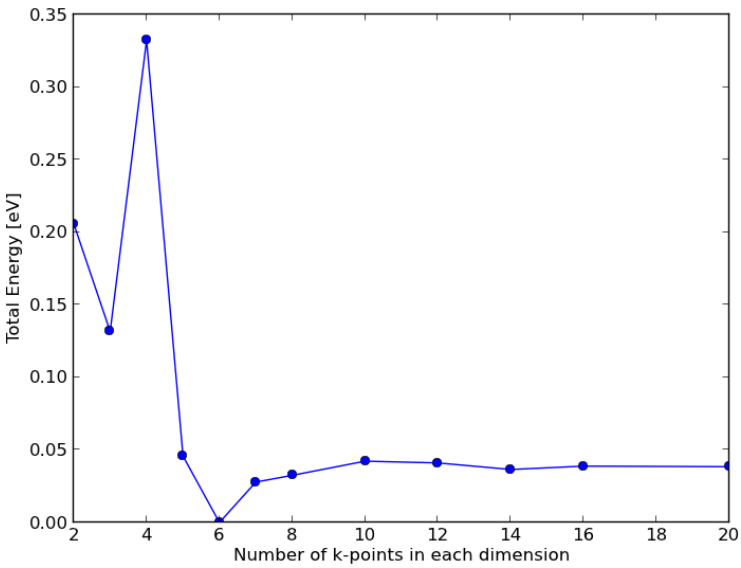

Figure 2: Convergence of k-point grid for Subproblem 1.2.

Problem 2

The code shown below computes the lattice constant for both fcc and bcc tantalum as well as the bulk modulus of bcc tantalum.

```
#!/usr/bin/env python

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

encut = 300
kpts = (6,6,6)

# fcc
LC = [3.5 + (i + 1)*0.1 for i in range(20)]
fcc_energies, fcc_volumes = [], []
ready = True
for a in LC:
    atoms = Atoms([Atom('Ta',(0, 0, 0))],
                  cell=0.5*a*np.array([[1.0, 1.0, 0.0],
                                       [0.0, 1.0, 1.0],
                                       [1.0, 0.0, 1.0]]))
    with jasp('bulk/hw04/Ta-fcc-{0}'.format(a),
              xc='PBE',
              encut=encut,
              kpts=kpts,
              atoms=atoms) as calc:
        try:
            fcc_energies.append(calc.get_atoms().get_potential_energy())
            fcc_volumes.append(calc.get_atoms().get_volume())
        except (VaspSubmitted, VaspQueued):
            ready = False

if ready:
    # Fit fcc results to EOS
    from ase.utils.eos import EquationOfState

    eos = EquationOfState(fcc_volumes,fcc_energies)
    v0, e0, B = eos.fit()
    eos.plot('images/Ta-fcc-eos.png')
    fcc_lattice_optimal = (4.0*v0)**(1./3.)
    print 'Ta fcc: Lattice Constant: {0} Ang'.format(fcc_lattice_optimal)

    # Construct bcc from fcc
    atoms = Atoms([Atom('Ta',(0, 0, 0))],
                  cell=0.5*fcc_lattice_optimal*np.array([[ 1.0,  1.0, -1.0],
                                                         [-1.0,  1.0,  1.0],
                                                         [ 1.0, -1.0,  1.0]]))
    bcc_lattice_approx = fcc_lattice_optimal*(11.8/atoms.get_volume())**(1./3.)
    print 'Ta bcc: Approximate Lattice Constant: {0} Ang'.format(bcc_lattice_approx)

    # Run DFT calculations on bcc
    LC = [bcc_lattice_approx - i*0.1 for i in range(10, -1, -1)]
    LC += [bcc_lattice_approx + (i + 1)*0.1 for i in range(10)]
    bcc_energies, bcc_volumes = [], []
    ready = True
    for a in LC:
        atoms = Atoms([Atom('Ta',(0, 0, 0))],
                      cell=0.5*a*np.array([[ 1.0,  1.0, -1.0],
                                           [-1.0,  1.0,  1.0],
                                           [ 1.0, -1.0,  1.0]]))
        with jasp('bulk/hw04/Ta-bcc-{0}'.format(a),
                  xc='PBE',
                  encut=encut,
                  kpts=kpts,
                  atoms=atoms) as calc:
```

```
63              try:
64                  bcc_energies.append(calc.get_atoms().get_potential_energy())
65                  bcc_volumes.append(calc.get_atoms().get_volume())
66              except (VaspSubmitted, VaspQueued):
67                  ready = False
68
69      if ready:
70      # Fit data to EOS
71          eos = EquationOfState(bcc_volumes,bcc_energies)
72          v0, e0, B = eos.fit()
73          eos.plot('images/Ta-bcc-eos.png')
74          print 'Ta bcc: Lattice Constant = {0} Ang'.format((2.*v0)**(1./3.))
75          print 'Ta bcc: Bulk Modulus = {0} GPa'.format(B*160.217)
```

Output:

```
Ta fcc: Lattice Constant: 4.20289660374 Ang
Ta bcc: Approximate Lattice Constant: 2.86838428403 Ang
Ta bcc: Lattice Constant = 3.31823177204 Ang
Ta bcc: Bulk Modulus = 289.284891291 GPa
```

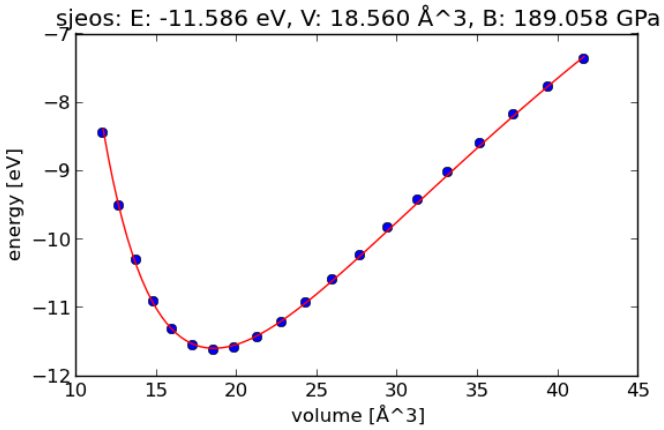

Figure 3: EOS fit for fcc tantalum for Problem 2.

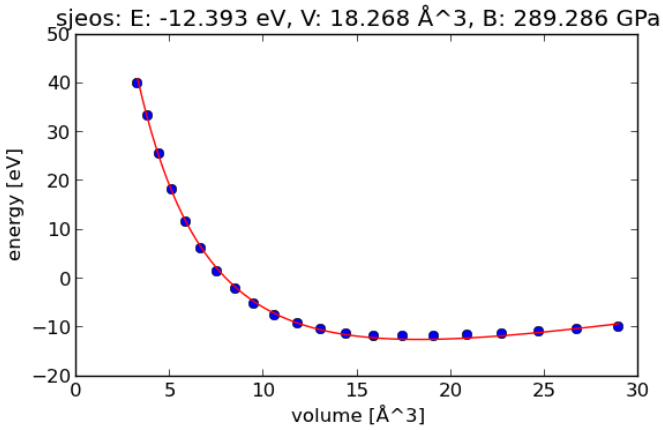

Figure 4: EOS fit for bcc tantalum for Problem 2.

From http://www.infoplease.com/periodictable.php?id=73, the experimental lattice constant of bcc Ta is 3.310 Å, which agrees well with the one obtained with DFT.

Problem 3

The code to compare the energies of fcc and bcc tantalum is shown below. All calculations were done in **Problem 2**. It can be concluded that bcc tantalum is more stable (more negative minimum of potential energy).

```
from jasp import *

# bcc energies and volumes
bcc_lattice_approx = 2.86838428403
bcc_LC = [bcc_lattice_approx - i*0.1 for i in range(1, -1, -1)]
bcc_LC += [bcc_lattice_approx + (i + 1)*0.1 for i in range(8)]
bcc_volumes, bcc_energies = [], []
for a in bcc_LC:
    with jasp('bulk/hw04/Ta-bcc-{0}' .format(a)) as calc:
        atoms = calc.get_atoms()
        bcc_volumes.append(atoms.get_volume())
        bcc_energies.append(atoms.get_potential_energy())

# fcc energies and volumes
fcc_LC = [3.5 + (i + 1)*0.1 for i in range(9)]
fcc_volumes, fcc_energies = [], []
for a in fcc_LC:
    with jasp('bulk/hw04/Ta-fcc-{0}'.format(a)) as calc:
        atoms = calc.get_atoms()
        fcc_volumes.append(atoms.get_volume())
        fcc_energies.append(atoms.get_potential_energy())

import matplotlib.pyplot as plt
plt.plot(fcc_volumes, fcc_energies, '-o', label='fcc')
plt.plot(bcc_volumes, bcc_energies, '-o', label='bcc')
plt.xlabel('Atomic Volume [$\AA^3$/atom]')
plt.ylabel('Total Energy [eV]')
plt.legend()
plt.savefig('images/Ta-bcc-fcc.png')
plt.show()
```

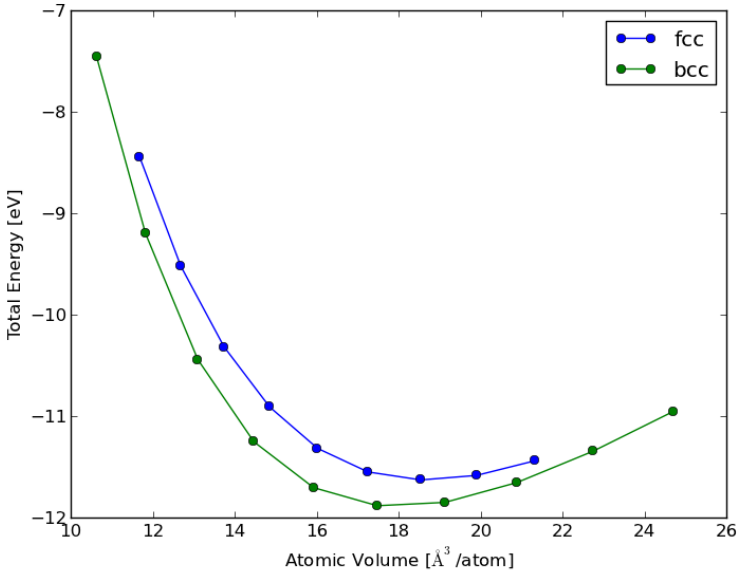

Figure 5: Energies for fcc and bcc tantalum for Problem 3.

Problem 4

Subproblem 4.1

The code and graph are shown below. The cut-off energy of 525 eV is sufficient for a convergence of 10 meV.

```
from ase.lattice.hexagonal import Graphite
from jasp import *

atoms = Graphite('C', latticeconstant={'a':2.4612, 'c':6.7079})
ENCUTS = [400, 425, 450, 475, 500, 525, 550]
TE = []

ready = True
for e in ENCUTS:
    with jasp('bulk/hw04/Graphite-encuts-{0}'.format(e),
              xc='PBE',
              kpts=(10, 10, 10),
              encut=e,
              atoms=atoms) as calc:
        try:
            TE.append(atoms.get_potential_energy())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

import matplotlib.pyplot as plt
TE = np.array(TE)
TE -= TE.min()
plt.plot(ENCUTS, TE, '-o')
plt.xlabel('Cut-off Energy [eV]')
plt.ylabel('Total Energy [eV]')
plt.savefig('images/Graphite-encut-convergence.png')
```

```
plt.show()
```

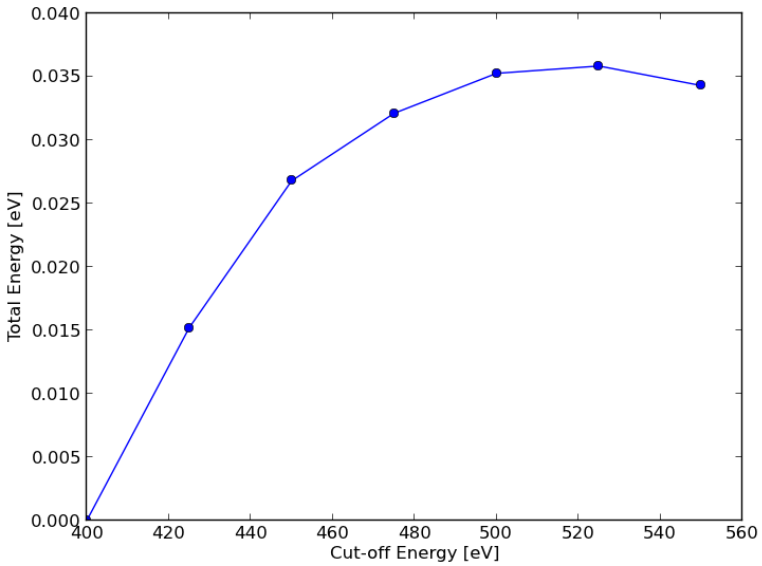

Figure 6: Convergence of energies for Subproblem 4.1.

Subproblem 4.2

The code and graph are shown below. A k-point grid of $8 \times 8 \times 8$ is sufficient to stabilize the calculations.

```
from ase.lattice.hexagonal import Graphite
from jasp import *

atoms = Graphite('C', latticeconstant={'a':2.4612, 'c':6.7079})
kpts = [2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20]
TE = []

ready = True
for k in kpts:
    with jasp('bulk/hw04/Graphite-kpts-{0}'.format(k),
              xc='PBE',
              kpts=(k, k, k),
              encut=525, # From part 1
              atoms=atoms) as calc:
        try:
            TE.append(atoms.get_potential_energy())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

import matplotlib.pyplot as plt
TE = np.array(TE)
TE -= TE.min()
plt.plot(kpts, TE, '-o')
plt.xlabel('Number of k-points in each dimension')
plt.ylabel('Total Energy [eV]')
plt.savefig('images/Graphite-kpts-convergence.png')
plt.show()
```

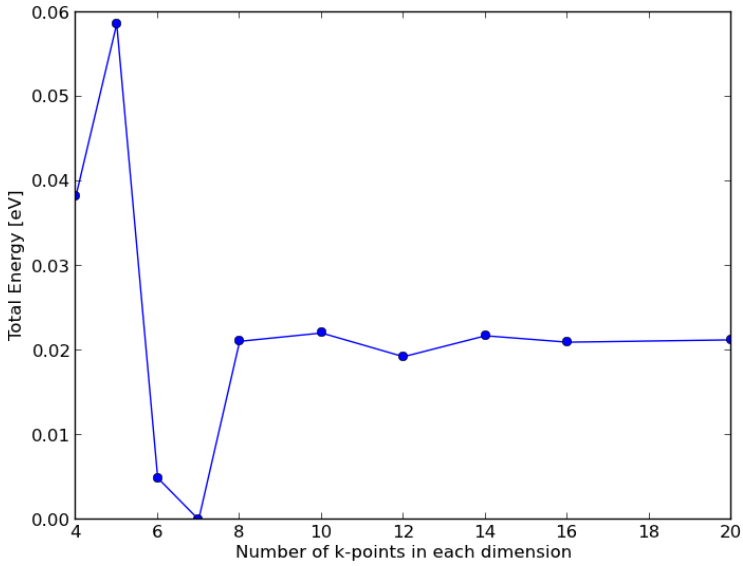

Figure 7: Convergence of k-point grid for Subproblem 4.2.

Subproblem 4.3

Using the values for energy cut-off and k-point grid obtained above, the code for geometry optimization of graphite is shown below. Only Step 1 (`ISIF = 2`) was used, since if we allow the volume to change (`ISIF = 4` and `ISIF = 3`), the sheets of graphite stop interacting with each other.

```
# Step 1: frozen atoms and shape at different volumes
from ase.lattice.hexagonal import Graphite
import numpy as np
from jasp import *
import matplotlib.pyplot as plt

atoms = Graphite('C', latticeconstant={'a':2.4612, 'c':6.7079})
encut = 525
kpts = (8,8,8)

v0 = atoms.get_volume()
cell0 = atoms.get_cell()

factors = [0.9, 0.95, 1.0, 1.05, 1.1] # To change volume by
energies, volumes = [], []
ready = True
for f in factors:
    v1 = f*v0
    cell_factor = (v1/v0)**(1./3.)
    atoms.set_cell(cell0*cell_factor, scale_atoms=True)
    with jasp('bulk/hw04/Graphite-step1-{0:1.2f}'.format(f),
        encut=encut,
        kpts=kpts,
        isif=2, # Relax internal degrees of freedom
        ibrion=1,
        nsw=50,
        xc='PBE',
```

```
        sigma=0.01,
        atoms=atoms) as calc:
        try:
            energies.append(atoms.get_potential_energy())
            volumes.append(atoms.get_volume())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

# Get volume factor corresponding to minimum energy
factors_min = factors[np.array(energies).argmin()]
print 'Volume factor of minimum energy = {0}'.format(factors_min)

with jasp('bulk/hw04/Graphite-step1-{0:1.2f}'.format(factors_min)) as calc:
  print calc

plt.plot(volumes, energies)
plt.xlabel('Vol. [$\AA^3$]')
plt.ylabel('Total energy [eV]')
plt.savefig('images/Graphite-step1.png')
plt.show()
```

Output:

```
Volume factor of minimum energy = 1.0
: ------------------------------
  VASP calculation from /home/bacalfa/Documents/Aptana_Studio_3_Workspace/06-640_HWs/bul
  converged: True
  Energy = -36.823508 eV

  Unit cell vectors (angstroms)
        x       y     z      length
  a0 [ 2.461  0.000  0.000] 2.461
  a1 [-1.231  2.131  0.000] 2.461
  a2 [ 0.000  0.000  6.708] 6.708
  a,b,c,alpha,beta,gamma (deg): 2.461 2.461 6.708 90.0 90.0 90.0
  Unit cell volume = 35.189 Ang^3
  Stress (GPa):xx,   yy,    zz,    yz,    xz,    xy
             -0.019 -0.019 -0.023 -0.000 -0.000 -0.000
 Atom#  sym       position [x,y,z]         tag  rmsForce constraints
   0    C    [0.000      0.000      0.000]   0   0.00      T T T
   1    C    [0.000      1.421      0.000]   0   0.00      T T T
   2    C    [0.000      1.421      3.354]   0   0.00      T T T
   3    C    [1.231      0.710      3.354]   0   0.00      T T T
--------------------------------------------------

INCAR Parameters:
-----------------
        nbands: 12
           nsw: 50
        ibrion: 1
```

```
          isif: 2
         encut: 525.0
         sigma: 0.01
       magmom: None
          prec: Normal
          kpts: [8, 8, 8]
    reciprocal: False
            xc: PBE
           txt: -
         gamma: False

Pseudopotentials used:
----------------------
C: potpaw_PBE/C/POTCAR (git-hash: 2272d6745da89a3d872983542cef1d18750fc952)
```

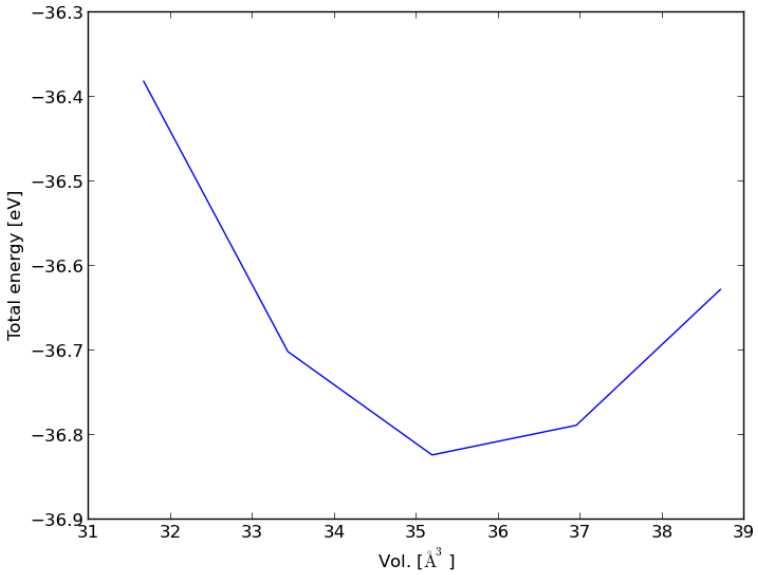

Figure 8: Geometry optimization of graphite (Step 1) for Subproblem 4.3.

Problem 5

The code below shows the execution of the 3 steps in the geometry optimization of TaC in B1 crystal structure. The lattice constant calculated was 4.48 Å and the bulk modulus was 337.27 GPa.

```
1 from ase.lattice.compounds import B1
2 from jasp import *
3 import matplotlib.pyplot as plt
4
5 atoms = B1(['Ta', 'C'], latticeconstant=3.169)
6 encut = 520
7 kpts = (8,8,8)
8
```

```
v0 = atoms.get_volume()
cell0 = atoms.get_cell()

factors = [2.3 + 0.1*i for i in range(14)] # To change volume by
energies, volumes = [], []
ready = True
for f in factors:
    v1 = f*v0
    cell_factor = (v1/v0)**(1./3.)
    atoms.set_cell(cell0*cell_factor, scale_atoms=True)
    with jasp('bulk/hw04/TaC-step1-{0:1.2f}'.format(f),
        encut=encut,
        kpts=kpts,
        isif=2, # Relax internal degrees of freedom
        ibrion=1,
        nsw=50,
        xc='PBE',
        sigma=0.01,
        atoms=atoms) as calc:
        try:
            energies.append(atoms.get_potential_energy())
            volumes.append(atoms.get_volume())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

plt.plot(volumes, energies, '-o')
plt.xlabel('Vol. [$\AA^3$]')
plt.ylabel('Total energy [eV]')
plt.savefig('images/TaC-step1.png')
plt.show()

# Step 2: allow unit cell shape to change, but at constant volume
energies1, volumes1 = [], [] # From Step 1
energies, volumes = [], [] # For Step 2
ready = True
for f in factors:
    with jasp('bulk/hw04/TaC-step1-{0:1.2f}'.format(f)) as calc:
        atoms = calc.get_atoms()
        energies1.append(atoms.get_potential_energy())
        volumes1.append(atoms.get_volume())
        calc.clone('bulk/hw04/TaC-step2-{0:1.2f}'.format(f))

    # Set ISIF = 4 and run
    with jasp('bulk/hw04/TaC-step2-{0:1.2f}'.format(f),
              isif=4) as calc:
        atoms = calc.get_atoms()
        try:
            energies.append(atoms.get_potential_energy())
            volumes.append(atoms.get_volume())
        except (VaspSubmitted, VaspQueued):
            ready = False

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

plt.plot(volumes1, energies1, volumes, energies)
plt.xlabel('Vol. [$\AA^3$]')
plt.ylabel('Total energy [eV]')
plt.legend(['Step 1', 'Step 2'], loc='best')
plt.savefig('images/TaC-step2.png')
plt.show()

# Get volume factor corresponding to minimum energy
factors_min = factors[np.array(energies).argmin()]
print 'Volume factor of minimum energy = {0}'.format(factors_min)
```

```
77
78 # Step 3: optimize unit cell volume, shape and internal coordinates
79 with jasp('bulk/hw04/TaC-step2-{0:1.2f}'.format(factors_min)) as calc:
80     calc.clone('bulk/hw04/TaC-step3')
81
82 # Set ISIF = 3, run, and print calculator
83 with jasp('bulk/hw04/TaC-step3',
84           isif=3) as calc:
85     try:
86         calc.calculate()
87         atoms = calc.get_atoms()
88         print calc
89     except (VaspSubmitted, VaspQueued):
90         pass
91
92 from ase.utils.eos import EquationOfState
93
94 eos = EquationOfState(volumes, energies)
95 v0, e0, B = eos.fit()
96 print 'Step 1'
97 print '======'
98 print '''
99 v0 = {0} A^3
100 E0 = {1} eV
101 B = {2} GPa
102 LC = {3} A'''.format(v0, e0, B*160.217, v0**(1./3.))
103 eos.plot('images/TaC-step2-eos.png')
```

Output:

```
Volume factor of minimum energy = 2.8
: ------------------------------
  VASP calculation from /home/bacalfa/Documents/Aptana_Studio_3_Workspace/06-640_HWs/bul
  converged: True
  Energy = -88.995309 eV

  Unit cell vectors (angstroms)
        x       y     z      length
  a0 [ 4.476  0.000  0.000] 4.476
  a1 [ 0.000  4.476  0.000] 4.476
  a2 [ 0.000  0.000  4.476] 4.476
  a,b,c,alpha,beta,gamma (deg): 4.476 4.476 4.476 90.0 90.0 90.0
  Unit cell volume = 89.649 Ang^3
  Stress (GPa):xx,   yy,    zz,    yz,    xz,    xy
            -0.474 -0.474 -0.474  0.000  0.000  0.000
 Atom#  sym       position [x,y,z]          tag  rmsForce constraints
   0    Ta  [0.000      0.000      0.000]   0   0.00      T T T
   1    C   [0.000      0.000      2.238]   0   0.00      T T T
   2    C   [0.000      2.238      0.000]   0   0.00      T T T
   3    Ta  [0.000      2.238      2.238]   0   0.00      T T T
   4    C   [2.238      0.000      0.000]   0   0.00      T T T
   5    Ta  [2.238      0.000      2.238]   0   0.00      T T T
   6    Ta  [2.238      2.238      0.000]   0   0.00      T T T
   7    C   [2.238      2.238      2.238]   0   0.00      T T T
```

```
------------------------------------------------

INCAR Parameters:
-----------------
        nbands: 23
           nsw: 50
        ibrion: 1
          isif: 3
         encut: 520.0
         sigma: 0.01
        magmom: None
          prec: Normal
          kpts: [8, 8, 8]
    reciprocal: False
            xc: PBE
           txt: -
         gamma: False

Pseudopotentials used:
----------------------
C: potpaw_PBE/C/POTCAR (git-hash: 2272d6745da89a3d872983542cef1d18750fc952)
Ta: potpaw_PBE/Ta/POTCAR (git-hash: e9b3f5148e6473afa92608154e25e8d5ca394b1a)
Step 1
======

v0 = 89.6537307425 A^3
E0 = -88.9942978103 eV
B = 337.270044991 GPa
LC = 4.47565005369 A
```

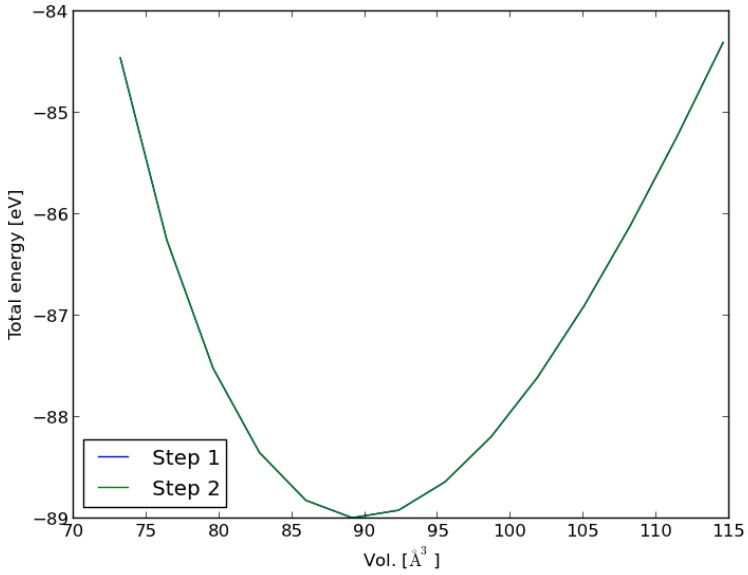

Figure 9: Steps 1 and 2 for Problem 5.

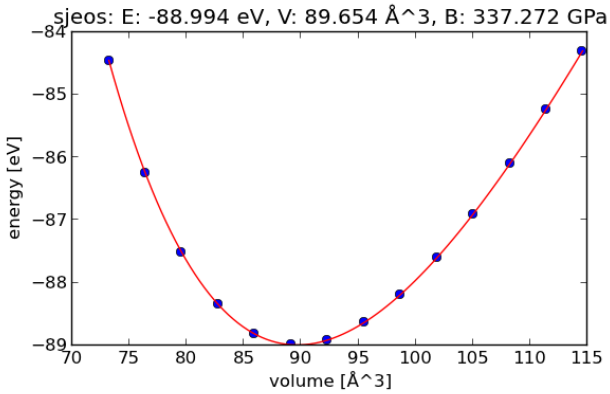

Figure 10: EOS for Step 2 for Problem 5.

Problem 6

The energy of formation of TaC is calculated to be $-1.17\,\text{eV}$ or $-26,971\,\text{cal/mol}$. See code below. It is not straightforward to compare that energy value with the expression contained in the paper, $\Delta F^\circ(\pm 800) = -34,900(\pm 700) + 0.5T$ cal/g.atom of Ta, because the range of T used to obtain that expression, [1,250, 1400] K, was very far from 0 K.

```
from jasp import *

eTaC, eTa, eC = None, None, None

with jasp('bulk/hw04/TaC-step3') as calc:
```

```
    atoms = calc.get_atoms()
    eTaC = atoms.get_potential_energy()/(len(atoms)/2.)

bcc_lattice_approx = 2.86838428403
bcc_LC = [bcc_lattice_approx - i*0.1 for i in range(1, -1, -1)]
bcc_LC += [bcc_lattice_approx + (i + 1)*0.1 for i in range(8)]
bcc_energies = []
for a in bcc_LC:
    with jasp('bulk/hw04/Ta-bcc-{0}' .format(a)) as calc:
        atoms = calc.get_atoms()
        bcc_energies.append(atoms.get_potential_energy())

eTa = min(bcc_energies);

factors_min = 1.0
with jasp('bulk/hw04/Graphite-step1-{0:1.2f}'.format(factors_min)) as calc:
    atoms = calc.get_atoms()
    eC = atoms.get_potential_energy()/len(atoms)

print 'Energy of formation of TaC = {0} eV'.format(eTaC - eTa - eC)
```

Output:

```
Energy of formation of TaC = -1.16988925 eV
```