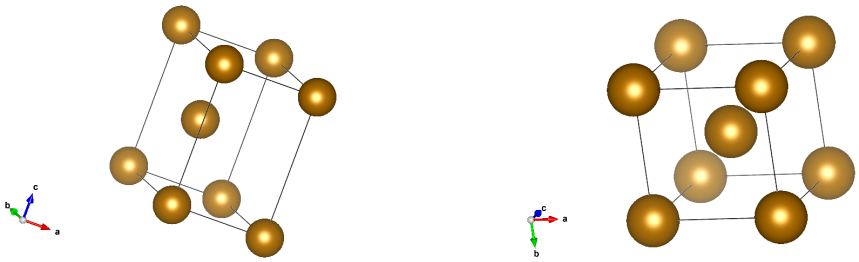
Computational Design of Materials Lab 3

Andrew Sullivan

March 28, 2019

1 Problem 1: Iron Stability Under Pressure and Magnetism

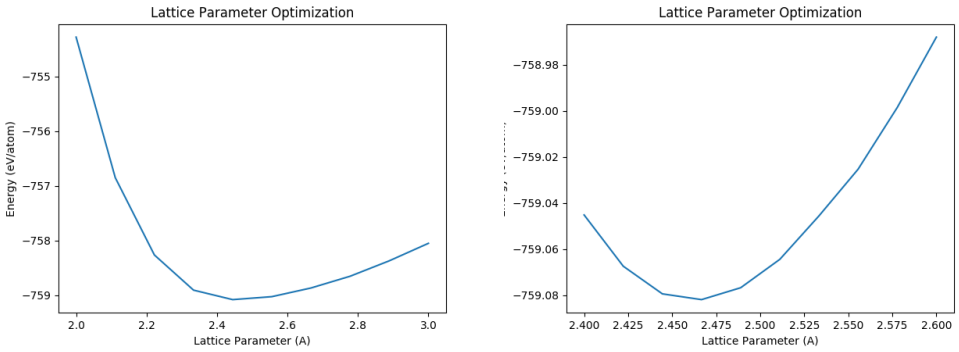
1.1 A: Lattice Parameter Optimization



(a) HCP. (b) BCC.

Figure 1: HCP and BCC Cells.

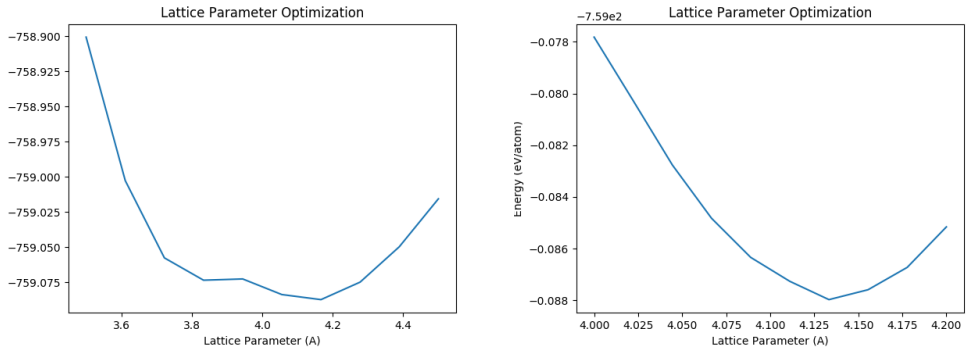
Figure 1 displays the constructed HCP and BCC cells for iron. It should be noted that the "cubic" parameter for the BCC structure in the "bulk" function used to create the cells is set to true to allow for specifying the two different atoms to have different identities for later magnetization calculations. The complete BCC unit cell, like the HCP primitive cell, has 2 atoms in it, thus allowing for direct comparison between energies even without finding the energy per atom.



(a) Coarse. (b) Fine.

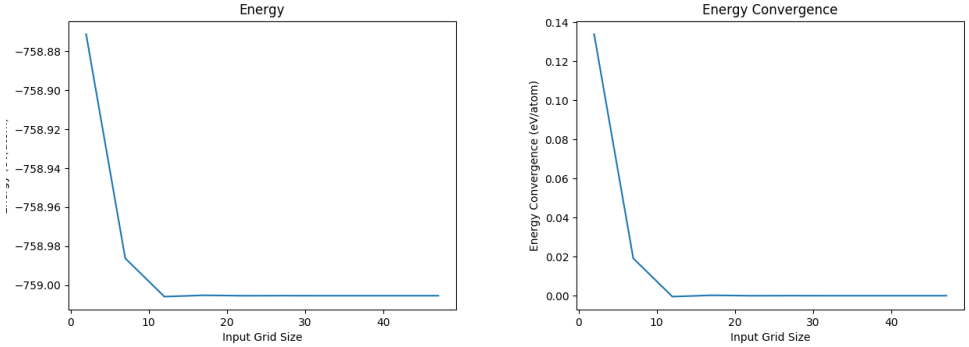
Figure 2: Optimization of HCP a parameter.

A similar procedure was used to optimize the lattice parameter(s) and k-point grid size for both HCP and BCC cells. Initially, the lattice parameter a was scanned over a range between 2.0 and 3.0 Å using the specified kinetic energy cutoff of 30 Ry and charge density cutoff of 300 Ry (so the ecutrho parameter was changed to be 10 times the ecut parameter) and with an initial k-point grid input of 4. For the HCP structure, the third k-point grid dimension was set to floor(nk/2) to account for differences between the a and c parameters. For the initial a optimization of the HCP structure, the c parameter was set to be $c = \sqrt{\frac{8}{3}}a$, the ideal value for the HCP structure. Figure 2**a** shows the results from this initial optimization. Using an estimated 2.45 Å as the ideal lattice parameter as an input, the c parameter was optimized using a similar technique, first testing between 3.5 and 4.5 Å to capture the ideal expected value. Using the minimum at approximately 4.15 Å (see Figure 3**a**), the k-point grid was then optimized as in Figure 4. As before, we scan over a range between 2 and 52 in increments of 5 as the input parameters, and plot both the energy and the convergence (defined as the difference between the energy at the largest k-point grid and the i^{th} grid). A converged energy is defined by one with a convergence smaller than 2 meV/atom. The energy was found



(a) Coarse. (b) Fine.

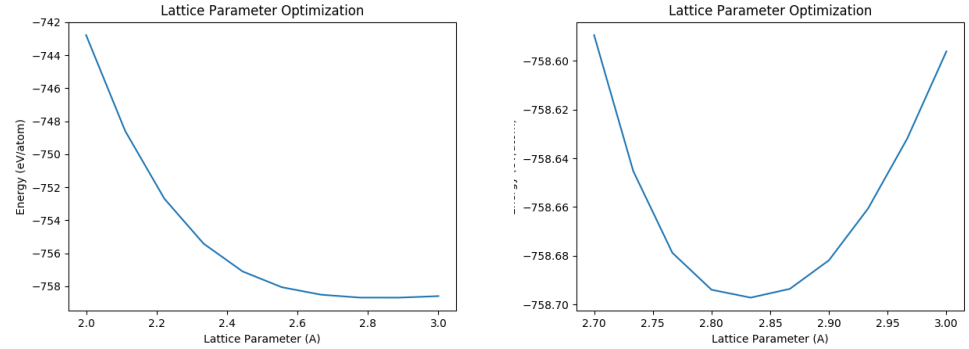
Figure 3: Optimization of HCP c parameter.



(a) Energy. (b) Convergence of Energy.

Figure 4: K-Point Convergence of HCP Energy.

to converge by an input grid size of 20, so this was used for further calculations. The lattice parameters a and c were then optimized with the corrected grid size over finer ranges, 2.4 to 2.6 Å for a and 4.0 to 4.2 Å for c, yielding final values of 2.47 Å and 4.13 Å, respectively (Figures **2a** and 3**a**). This yields a c/a ratio of 1.67, quite close to the ideal value of 1.63.

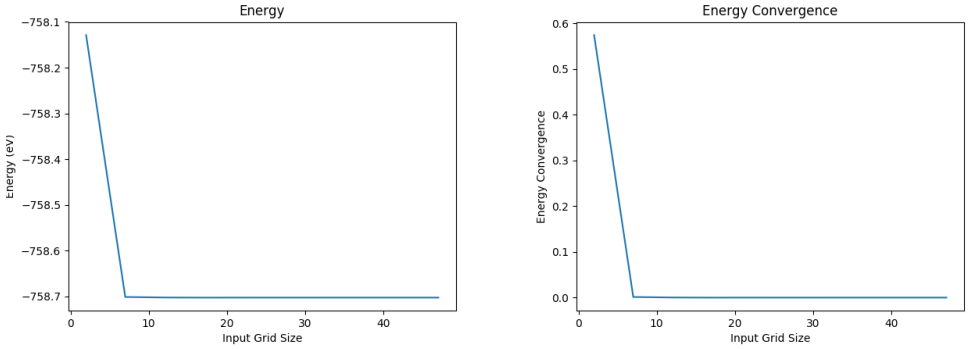


(a) Coarse. (b) Fine.

Figure 5: Optimization of BCC a parameter.

As discussed above, a nearly identical procedure was used for BCC, but no second lattice parameter had

to be optimized due to the cubic symmetry of the lattice. An initial range of 2.0 to 3.0 was tested to start (Figure 5**a**), and an input parameter of 2.80 Å was used to optimize the k-point grid (Figure 6). Once the grid was optimized, which again occurred by an input k-point parameter of 20, a finer range between 2.8 and 2.9 Å was tested to locate the ideal value of 2.83 Å.



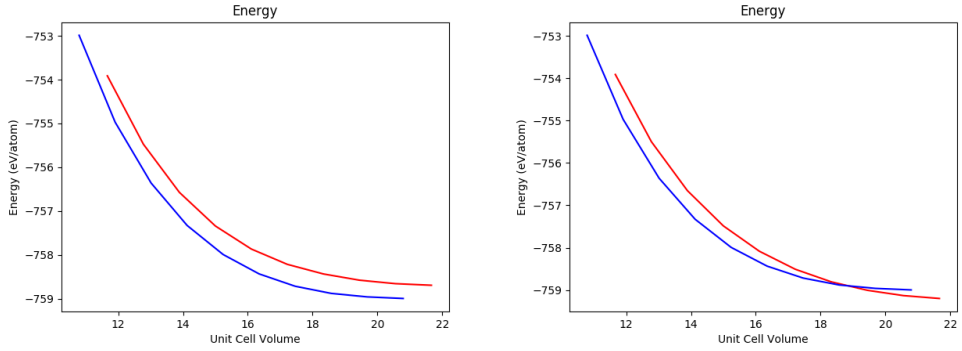
(a) Energy.

(b) Convergence of Energy.

Figure 6: K-Point Convergence of BCC Energy.

It should be noted that, for all calculations in the above section, the "compute energy anti" function was used to initialize an antiferromagnetic structure by declaring two atom types (named "Fe" and "Co" but both set to the Fe pseudopotential) and initializing one class with a starting magnetization of 1 and the other with -1. As found in the next two parts, the energies will vary for different magnetizations, so absolute energies can only be directly compared between antiferromagnetic (AFM) HCP and BCC Fe in this part, not between any arbitrary HCP and BCC Fe structure.

1.2 B: Stability of HCP vs. BCC Structures



(a) AFM Starting Configurations.

(b) AFM for HCP and FM for BCC.

Figure 7: Energy vs. Unit Cell Volume for Starting Magnetic Configurations. Blue = HCP. Red = BCC.

To examine the transition between BCC and HCP Fe at high pressures, we strain each lattice by altering the input lattice parameter and calculate the energy as a function of the new unit cell volume. Only positive pressures (i.e. the cell parameters decrease) are considered here, but the results suggest that there would be no further transition for pressures which stretch the lattice between these two phases. K-point grid inputs of 20 and kinetic energy cutoffs of 30 are used for all calculations, along with the above lattice parameters of 2.83 for BCC and 2.47 for HCP, and a c/a ratio of 1.67. In straining our HCP lattice, it is assumed that the strain is isotropic such that the c/a parameter is fixed. We scan over a range of volume changes between

1 and 10 cubic Å in increments of 1. For each volume change dV, we calculate the new unit cell volume as $V - dV$, where V is found from the unit cell geometry. For the BCC structure, $V = a^3$. For a typical HCP unit cell (with 6 atoms), the volume is that of a hexagonal prism: $V = \frac{3\sqrt{3}}{2}a^2c$, so for our primitive lattice, which is 1/3 the volume of the whole cell, $V = \frac{\sqrt{3}}{2}a^2c = \frac{1.67\sqrt{3}}{2}a^3$ using our relation between the lattice parameters. From our new volume, $V - dV$, we can solve the two above equations for the new lattice parameter that will yield that volume, $a = V^{1/3}$ for BCC, and $a = (\frac{2V}{1.67\sqrt{3}})^{1/3}$ for HCP. We then compute the energy of the strained lattices by passing these as our inputs to the structure calculation, and plot the results as a function of $V - dV$.

Given that the BCC lattice has different magnetization states (see the next section) which have different energies, it is possible that the observed transition at 15 GPa between BCC and HCP is only the result of one magnetization state for each structure. If we initialize both phases with AFM magnetizations, we find that a transition does not seem likely to occur since the energy of the HCP phase is always lower than the BCC. Testing a number of different initial magnetizations for both BCC and HCP structures, we find that a transition does occur when HCP is initialized as AFM and BCC is initialized as ferromagnetic (FM). This is consistent with the literature, which shows that these states are the ground state magnetizations for both structures. In this case, we find that the transition occurs at a unit cell volume of approximately 19 cubic Å. Using our ideal lattice parameters, we find ground-state (unstrained) unit cell volumes of 22.7 and 21.8 cubic Å for BCC and HCP, respectively, suggesting volume changes of 3.7 and 2.8 cubic Å result in sufficient energy changes to induce the transformation.

By looking at the output files, we find that the energies are indeed equal at the closest values to the calculated lattice parameters for a cell volume of 19. Using these closest values and their nearest neighbors, we can approximate the pressure, which is proportional to the derivative of energy with respect to volume, using a second-order finite difference approximation:

$$P = -\frac{\partial E}{\partial V} \approx -\frac{E(V+h) + E(V-h) - 2E(V)}{h^2}$$

As mentioned above, using our range with the appropriate number of divisions, we have $h = 1.11$ cubic Å, and for the BCC structure, our three energies are -111.57183685, -111.49892289, and -111.54256674 Ry. This yields a pressure of 25 GPa, somewhat larger than the expected value of 15 GPa. Performing the same calculation for the HCP structure yields a pressure of 12 GPa, quite close to the expected transition pressure. It seems likely that these discrepancies are a consequence of the numerical approximation to the derivative. A finer grid would more closely capture the instantaneous slope and give a better approximation to the true transition pressure, especially for the BCC case, though it should also be noted that we do not require equality of pressure at the transition, since the volume and energy changes are also unequal, and the slopes of the two curves are clearly different even when their energies are equal.

1.3 C: Different Magnetic States of BCC Fe

In order to examine the effect of magnetization on stability of BCC Fe, we perform three calculations that differ only in their initial magnetization. All use a grid size of 20 x 20 x 20, a cutoff of 30 Ry, and BCC structure. For one calculation, we set the "anti" parameter to be true so that the two spins are initialized with starting magnetizations of -1 and 1. For the other two, this parameter is set to false. For one, we set the starting magnetization of both groups to be 0.7, and to 0.0 for the other, corresponding to ferromagnetic and non-magnetic states, respectively. For all three structures, we compute the energy as a function of lattice parameter and plot the results below. The AFM configuration is the red curve, the non-magnetic is in blue, and the ferromagnetic state is green. As expected from the literature, the ferromagnetic state has the lowest energy and is thus the most stable.

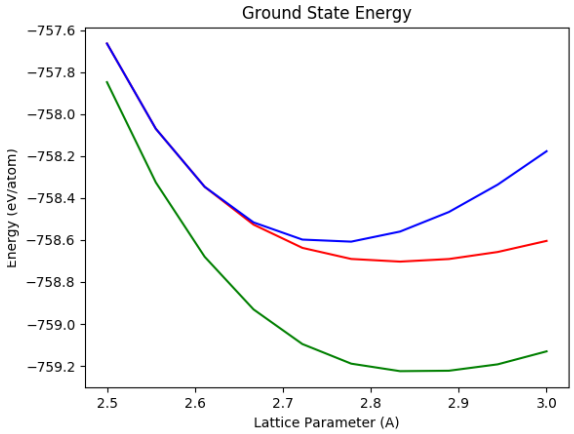


Figure 8: Energies of Different Magnetization States of BCC Fe. Blue = NM. Red. = AFM. Green = FM.

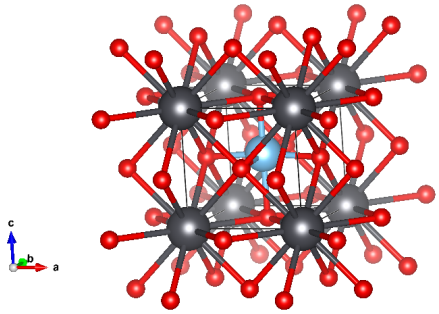
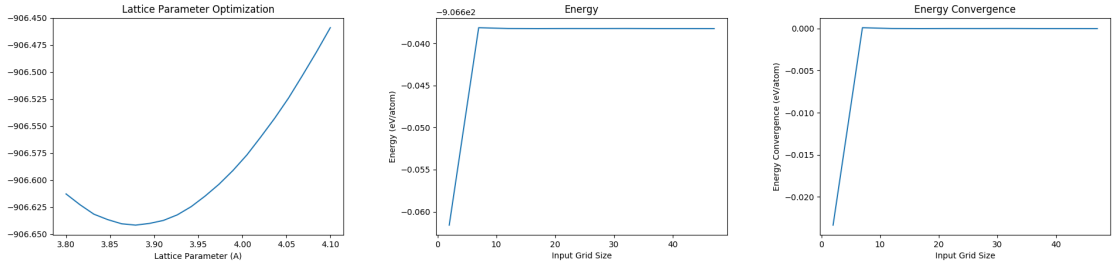


Figure 9: Perovskite Structure.



(a) Parameter Optimization. (b) Energy. (c) Convergence of Energy.

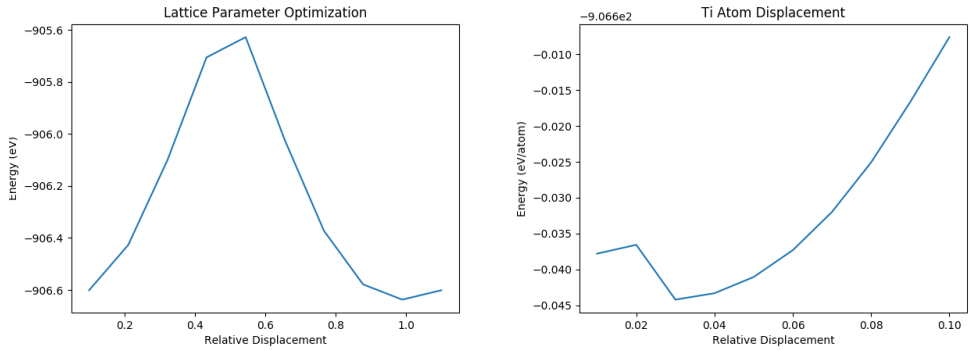
Figure 10: Parameter Optimization and K-Point Convergence of Lead Titanate Energy.

2 Problem 2: Stability of the Perovskite Lead Titanate

2.1 A: Lattice Parameter Optimization

To examine the perovskite material lead titanate, $PbTiO_3$ (pictured above), we begin with a fixed grid of 4 x 4 x 4 and an offset of 1, 1, 1 and sample lattice parameters in the range 3.8 to 4.1 Å using increments of 0.03 Å. This range is chosen to include the room-temperature experimental constant of 3.97 Å. Using this method, we find the minimum energy at a lattice parameter of 3.88 Å as shown above. K-point convergence is then evaluated using the same increments and range as those in the previous problem. We find that the grid

converges rather quickly, by an input size of 7 x 7 x 7, which is used in subsequent parts. This configuration, which does not allow relaxation of any atoms, has no net force and a total energy of -333.18323035 Ry.



(a) Coarse. (b) Fine.

Figure 11: Energy vs. Displacement of Ti Atom.

2.2 B: Displacement of Ti Atom

With our optimized lattice parameter, we then examine the energy as a function of relative displacement (in terms of the unit cell parameter). The perovskite structure depicted above shows a small displacement along the c direction as an example. Using constraints on both the Pb and Ti atoms, we relax the cell, allowing the O atoms to move. We initially scan over the entire range between 0.1 and 1.0 to try and pinpoint the region where the minimum may be, but find that it appears to be within a relative displacement less than 0.1. We then narrow our calculation range to 0.01 to 0.1 in increments of 0.01, finding a minimum at 0.03. This new structure has a total energy of -333.18515520 Ry, lower than that of the undisplaced configuration, suggesting the favorable energetics of the titanium displacement. All of the relaxed oxygen atoms shift their locations in the z direction in the same direction as the titanium atom. While x and y coordinates remain unchanged, the oxygen atoms with z of $0.5a$ to begin with shift to $0.577a$, and the oxygen with an initial z of 0 shifts to $0.063a$. Unlike the unrelaxed case, all the atoms have forces acting on them in the z direction, with the only positive force acting on the Pb.

2.3 C: Relaxation of Ti and O Atoms

Using a displacement of 0.03, we run the relax calculation again, with all the same inputs (nk = 7, ecut = 30, and the same lattice parameter), but remove the constraints on the Ti atom, allowing it to relax in addition to the O atoms. This relaxation shifts the energy to -333.18323515 Ry, lower than the cubic configuration but higher than the unrelaxed titanium condition. In this configuration, the oxygen with an initial z coordinate of 0 shifts by only $0.0024a$, and the displacements of the other oxygens and the titanium are similarly smaller than in the previous case. The other two oxygens shift in the $+z$ direction by $0.0023a$, while the titanium instead relaxes in the $-z$ direction by $0.0016a$, despite its initial displacement in the other direction. As in the previous case, we have forces acting on all atoms in the z direction, with the total force being two orders of magnitude larger than the previous case (0.0013 Ry/au). Here, the Pb and the third O atoms have negative forces, while the other three atoms have positive forces.

2.4 D: Stability of $PbTiO_3$ Phases

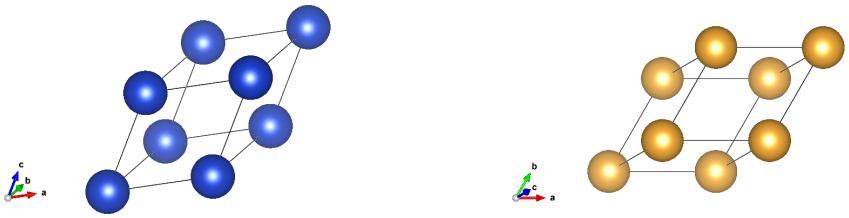
We can compare the energies of the minimized lattice parameter in **A**, the O-relaxed displaced cell in **B**, and the O- and Ti-relaxed displaced cell in **C** to determine the most stable configuration. To summarize, the unrelaxed condition produced the largest (least negative) energy, suggesting it is the least stable configuration, while the case where only the oxygen atoms were allowed to relax produced the most energetically

favorable configuration. The final case (wherein titanium was also allowed to relax) did not produce a more energetically stable configuration, but rather one with larger total energy and total force. The presence of non-zero forces suggests that these states are non-equilibrium, however, which is consistent with the fact that the unit cell was not allowed to relax (vc-relax) and that the lead atoms were fixed in place. Forbidding the optimization of lattice parameters would prevent a true tetragonal structure from ever being produced, but the presence and direction of forces in the latter configuration suggests that allowing these relaxations would produce a non-cubic structure. The only atoms with negative forces in the third part are those located at the bottom of the cell, while all others seek to be displaced upwards. This upward force would further shift the oxygen atoms away from their initial position, suggesting an elongation of the cell in the z direction, consistent with a transformation to a tetragonal structure.

In accordance with this conclusion, running the simulation with the same initial lattice parameter, k-point grid, energy cutoff, and titanium displacement conditions but without any constraints on relaxation (i.e. changing the calculation to vc-relax and removing the constraints on the O, Ti, and Pb atoms) produces a structure with equal a and b lattice parameters but a larger c parameter (i.e. tetragonal). This produces an energy lower than any of the previous calculations (-333.1861 Ry), suggesting it is more energetically favorable than the cubic or non-optimized tetragonal structures. A small decrease in the a parameter to 3.85 Å is also observed, and, as expected, the relative coordinates for all atoms along the x and y directions are unchanged (all are still combinations of 0 and 0.5). The c parameter is larger than the other lattice parameter, suggesting an elongation during the tetragonal transition to 3.95 Å. In the final state, we find that the titanium atom remains at a relative displacement of approximately 0.03, consistent with our previous calculations. The two oxygen atoms with non-zero initial z coordinates are shifted by the same relative amount in the opposite direction. The Pb atom is shifted slightly upwards, while the third oxygen is shifted slightly downwards from their initial positions. This configuration is consistent with the ferroelectric state of this perovskite at zero temperature, as the positive central titanium atom is shifted away from the negative oxygen cage, disrupting the normal charge distribution and allowing the material to support a non-zero electric polarization.

3 Problem 3: Stability of the CuAu Alloy

3.1 A: Lattice Parameter Optimization of Cu and Au

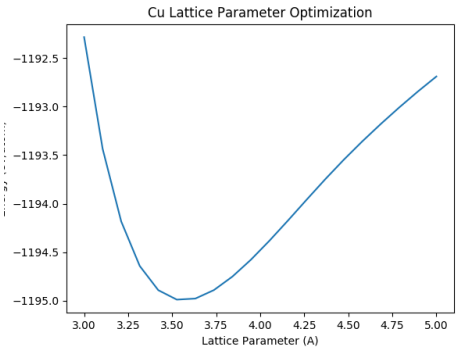


(a) Cu. (b) Au.

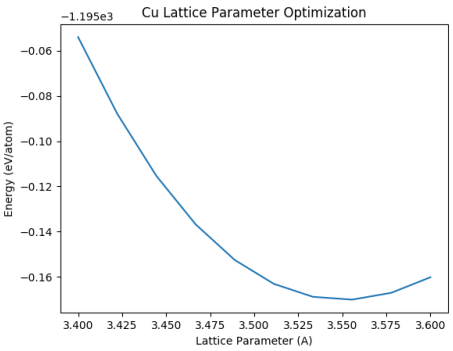
Figure 12: Cu and Au FCC Cells.

The above figure shows the primitive cells constructed for FCC Cu and Au. As these are primitive cells from an FCC lattice, each volume contains the equivalent of only one atom, so all energies are implicitly in eV/atom.

As above, we begin the optimization by searching over a grid with coarse spacing (here, between 3.0 and 5.0 in 0.1 Å steps). Using the minimal result from this calculation, we then optimize the k-point grid, and use this new input grid value in our final calculation for the optimized lattice parameter. For Cu, we find a final lattice parameter of 3.56 Å at a k-point grid of 15 x 15 x 15. In this case, we define convergence as an energy difference between the i^{th} grid and the final grid as smaller than 0.30 mHa/atom, or 0.008 eV/atom.

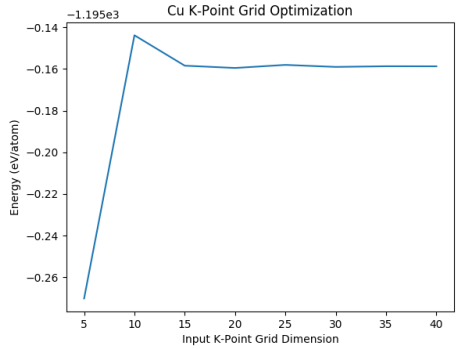


(a) Coarse.

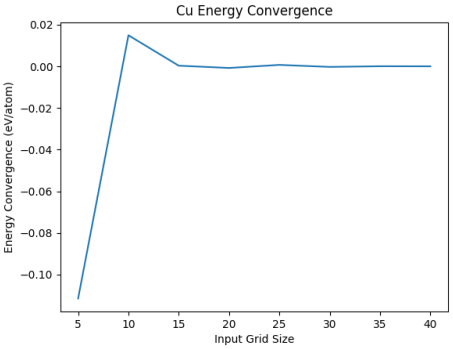


(b) Fine.

Figure 13: Optimization of Cu a parameter.

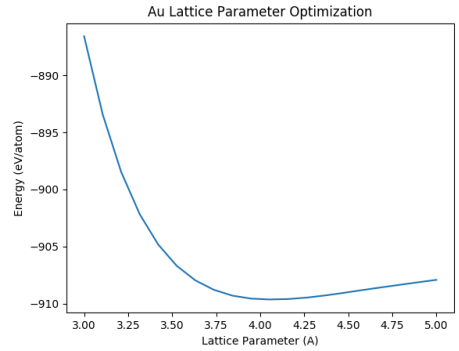


(a) Energy.

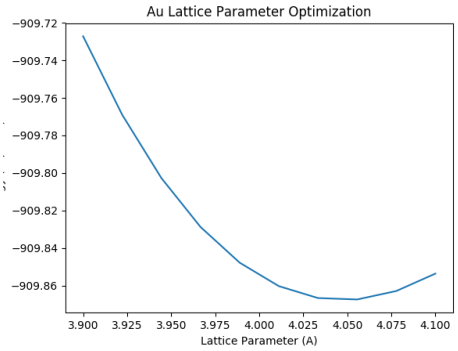


(b) Convergence of Energy.

Figure 14: K-Point Convergence of FCC Cu Energy.



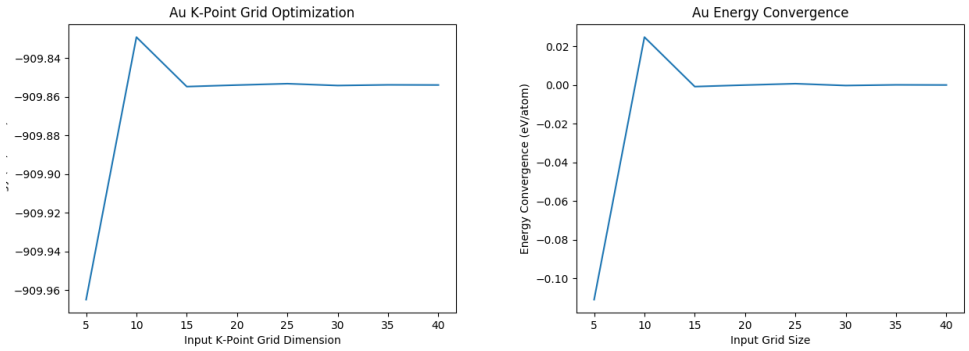
(a) Coarse.



(b) Fine.

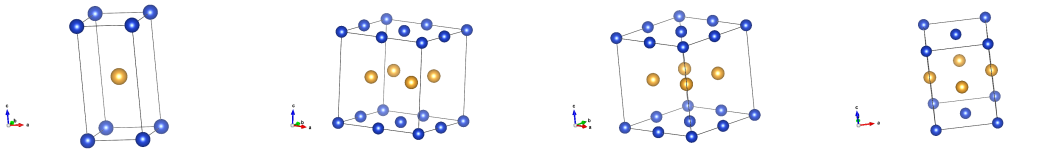
Figure 15: Optimization of Au a parameter.

We repeat this procedure for FCC Au and find a lattice parameter of 4.06 Å and that this structure has also converged by a 15 x 15 x 15 k-point grid.



(a) Energy. (b) Convergence of Energy.

Figure 16: K-Point Convergence of FCC Au Energy.



(a) BCT Cell. (b) BCT Supercell. (c) BCT Supercell. (d) PFCT Cell.

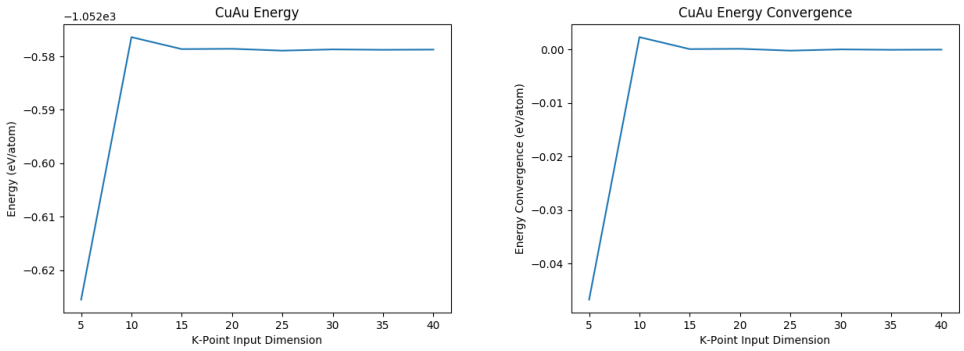
Figure 17: Equivalence of BCT and PFCT CuAu Cells.

3.2 B: Unit Cells of the $L1_0$ Phase of CuAu

In order to show the equivalence of the body-centered tetragonal (BCT) and pseudo-face-centered tetragonal (PFCT) cells of the CuAu alloy, we begin by constructing a single unit cell of the BCT structure with lattice parameters a, a, and c (subfigure **a**). We then construct a 2 x 2 x 1 supercell of this structure, shown in part **b** of the above figure, and in part **c** from a different view. From the view in part **c**, we can observe the PFCT unit cell embedded within this BCT supercell. The edges of the PFCT cell are formed by connecting the atoms at the center of each edge along the top, bottom, and side faces of the cell. The new faces formed by these edges will have the central atoms embedded in the center of the faces, rather than the center of the cell as in the initial BCT structure. This extracted structure is shown in the final subfigure (**d**) above. Clearly, the c parameter has not changed between the BCT and PFCT structures. The a parameter, however, is not equal to its initial value. Since we began by forming a supercell, the "lattice" parameter for the supercell in subfigures **b** and **c** is equal to $2a$. When we construct the PFCT cell embedded within this supercell, we form our new edges to connect the centers of each edge along the top and bottom faces. These edge-center atoms are located at the points $(a, 0, 0)$, $(0, a, 0)$, $(2a, a, 0)$, $(a, 2a, 0)$, and the same four locations on the top face (i.e. (x, y, c)). Thus, connecting any two pairs of these points that are *i)* both on the top or both on the bottom face, and *ii)* on edges that intersect at one of the corners/are not parallel requires construction of a line with length $a_{new} = \sqrt{a^2 + a^2} = \sqrt{2a^2}$.

3.3 C: Formation Energy of the CuAu Alloy

To optimize the BCT structure, we turn on vc-relax, start with nk = 4, ecut = 40, and set the initial structure as nearly cubic with the lattice parameter equal to average of two species' FCC parameters (3.81 Å) and the c parameter equal to 3.80 Å. We avoid making these equal since this may cause cubic symmetry to be included as a constraint, which would prevent us from finding the tetragonal state. An initial test with c = 2a was too slow, so the new starting value was used closer to the initial parameter. Ultimately, this yields lattice parameters of $a = 2.67$ Å and $c = 3.92$ Å. This a lattice parameter, as discussed above, corresponds to a PFCT lattice parameter of $a_{new} = \sqrt{2a^2} = 3.78$, close to the average between the two individual FCC parameters. The c/a ratio of this calculated cell is 1.04, and the c parameter is also close



(a) Energy.

(b) Convergence of Energy.

Figure 18: K-Point Convergence of BCT CuAu Energy.

to the average of the two FCC parameters, but is closer to the Au, while the a parameter is shifted towards the lattice constant for FCC Cu.

To evaluate the stability of this alloy, we compute its formation energy based on the energies of each constituent. To do so, we take the energy of the primitive FCC cell for Cu and of the primitive FCC cell for Au, both of which contain one atom, and subtract this energy from the computed energy of BCT CuAu, which similarly contains one of each type of atom. The total formation energy is then half of this difference:

$$\Delta H_f(CuAu) = \frac{1}{2}(E^{tot}(CuAu) - E^b(Cu) - E^b(Au))$$

This returns a formation energy of -60.3 meV, implying that the CuAu alloy is stable at 0K.

4 Appendix

The following function was used to perform all calculations for Problem 1, which were controlled by setting the P parameter to values between 1 and 6. The first section (P=1) is used to compute the a parameter for HCP or BCC cells (the Xtal parameter controls this, and clat is compared to alat to determine which type of cell is being used). P=2 is used to optimize c for HCP lattices with input a. P=3 and P=4 are used for k-point grid optimization for the two structures. P=5 computes energy as a function of unit cell volume for both structures and plots the result. Finally, P=6 performs lattice parameter vs. energy calculations for all three starting magnetic structures.

```
    from my_labutil.src.plugins.pwscf import *
from ase.spacegroup import crystal
from ase.io import write
from ase.build import bulk
import numpy
import matplotlib.pyplot as plt

def make_struc(alat,clat):
    """
    Creates the crystal structure using ASE.
    :param alat: Lattice parameter in angstrom
    :return: structure object converted from ase
    """
    if alat != clat:
        fecell = bulk('Fe', 'hcp', a=alat, c=clat)
        write('fehcp.cif', fecell)
    else:
        fecell = bulk('Fe', 'bcc', a=alat, cubic=True)
        write('febcc.cif', fecell)
    # check how your cell looks like

    print(fecell, fecell.get_atomic_numbers())
    fecell.set_atomic_numbers([26, 27])
    structure = Struc(ase2struc(fecell))
    print(structure.species)
    return structure


def compute_energy(alat, nk, ecut, clat, SM):
    """
    Make an input template and select potential and structure, and the path where to run
    """
    potname = 'Fe.pbe-nd-rrkjus.UPF'
    potpath = os.path.join(os.environ['ESPRESSO_PSEUDO'], potname)
    pseudopots = {'Fe': PseudoPotential(path=potpath, ptype='uspp', element='Fe',
                                        functional='GGA', name=potname),
                  'Co': PseudoPotential(path=potpath, ptype='uspp', element='Fe',
                                        functional='GGA', name=potname)
                  }
    struc = make_struc(alat=alat,clat=clat)
    if clat != alat:
        kpts = Kpoints(gridsize=[nk, nk, int(numpy.floor(nk/2))], option='automatic', offset
    =False)
    else:
        kpts = Kpoints(gridsize=[nk, nk, nk], option='automatic', offset=False)
    dirname = 'Fe_c_{}_ecut_{}_nk_{}_SM_{}'.format(clat, ecut, nk, SM)
    runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab3/Problem1/BCC/Mag", dirname)
    )
    input_params = PWscf_inparam({
        'CONTROL': {
            'calculation': 'scf',
            'pseudo_dir': os.environ['ESPRESSO_PSEUDO'],
            'outdir': runpath.path,
            'tstress': True,
            'tprnfor': True,
            'disk_io': 'none',
        },
```

```
        'SYSTEM': {
            'ecutwfc': ecut,
            'ecutrho': ecut * 10,
            'nspin': 2,
            'starting_magnetization(1)': SM,
            'starting_magnetization(2)': SM,
            'occupations': 'smearing',
            'smearing': 'mp',
            'degauss': 0.02
             },
        'ELECTRONS': {
            'diagonalization': 'david',
            'mixing_beta': 0.5,
            'conv_thr': 1e-7,
        },
        'IONS': {},
        'CELL': {},
        })

    output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
                               params=input_params, kpoints=kpts, ncpu=1, nkpool=1)
    output = parse_qe_pwscf_output(outfile=output_file)
    return output['energy']

def compute_energy_anti(alat, nk, ecut, clat):
    """
    Make an input template and select potential and structure, and the path where to run
    """
    potname = 'Fe.pbe-nd-rrkjus.UPF'
    potpath = os.path.join(os.environ['ESPRESSO_PSEUDO'], potname)
    pseudopots = {'Fe': PseudoPotential(path=potpath, ptype='uspp', element='Fe',
                                        functional='GGA', name=potname),
                  'Co': PseudoPotential(path=potpath, ptype='uspp', element='Fe',
                                        functional='GGA', name=potname)
                  }
    struc = make_struc(alat=alat,clat=clat)
    if clat != alat:
        kpts = Kpoints(gridsize=[nk, nk, int(numpy.floor(nk/2))], option='automatic', offset
    =False)
    else:
        kpts = Kpoints(gridsize=[nk, nk, nk], option='automatic', offset=False)
    dirname = 'Fe_a_{}_ecut_{}_nk_{}'.format(alat, ecut, nk)
    runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab3/Problem1/JUNK", dirname))
    input_params = PWscf_inparam({
        'CONTROL': {
            'calculation': 'scf',
            'pseudo_dir': os.environ['ESPRESSO_PSEUDO'],
            'outdir': runpath.path,
            'tstress': True,
            'tprnfor': True,
            'disk_io': 'none',
        },
        'SYSTEM': {
            'ecutwfc': ecut,
            'ecutrho': ecut * 10,
            'nspin': 2,
            'starting_magnetization(1)': 1,
            'starting_magnetization(2)': -1,
            'occupations': 'smearing',
            'smearing': 'mp',
            'degauss': 0.02
             },
        'ELECTRONS': {
            'diagonalization': 'david',
            'mixing_beta': 0.5,
            'conv_thr': 1e-7,
        },
        'IONS': {},
```

```
        'CELL': {},
        })

    output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
                               params=input_params, kpoints=kpts, ncpu=1, nkpool=1)
    output = parse_qe_pwscf_output(outfile=output_file)
    return output['energy']


def lattice_scan(nk,ecut,alat,clat,anti,SM):
    if anti:
        output = compute_energy_anti(alat=alat, ecut=ecut, nk=nk, clat=clat)
    else:
        output = compute_energy(alat=alat, ecut=ecut, nk=nk, clat=clat,SM=SM)
    print(output)
    return output

if __name__ == '__main__':
    # put here the function that you actually want to run
    P=5
    Q=1
    #Xtal set to 1 for HCP and 2 for BCC
    if Q == 1:
        E=[]
        if P == 1:
            Xtal=2
            for alat in numpy.linspace(2.8,2.9,5):
                if Xtal == 2:
                    clat=alat
                elif Xtal == 1:
                    clat = alat*(8/3)**(1/2)
                E.append(lattice_scan(nk=20,ecut=30,alat=alat, clat=clat)/2)
            plt.plot(numpy.linspace(2.8,2.9,5), E)
            plt.xlabel('Lattice Parameter (A)')
            plt.ylabel('Energy (eV/atom)')
            axes = plt.gca()
            plt.title("Lattice Parameter Optimization")
            plt.show()
        elif P == 2:
            alat=2.47
            for clat in numpy.linspace(4.0,4.2,10):
                E.append(lattice_scan(nk=4,ecut=30,alat=alat,clat=clat)/2)
            plt.plot(numpy.linspace(4.0,4.2,10), E)
            plt.xlabel('Lattice Parameter (A)')
            plt.ylabel('Energy (eV/atom)')
            axes = plt.gca()
            plt.title("Lattice Parameter Optimization")
            plt.show()
        elif P == 3:
            E=[]
            Ediff=[]
            alat = 2.47
            clat = 4.13
            for nk in range(2,52,5):
                E.append(lattice_scan(nk=nk,ecut=30,alat=alat,clat=clat)/2)
            for i in range(1,len(E)+1):
                Ediff.append(E[i-1]-E[len(E)-1])
            plt.plot(range(2,52,5), E)
            plt.xlabel('Input Grid Size')
            plt.ylabel('Energy (eV/atom)')
            axes = plt.gca()
            plt.title("Energy")
            plt.show()

            plt.plot(range(2, 52, 5), Ediff)
            plt.xlabel('Input Grid Size')
            plt.ylabel('Energy Convergence (eV/atom)')
            axes = plt.gca()
```

```
191             plt.title("Energy Convergence")
192             plt.show()
193         elif P == 4:
194             E=[]
195             Ediff=[]
196             alat = 2.83
197             clat=alat
198             for nk in range(2,52,5):
199                 E.append(lattice_scan(nk=nk,ecut=30,alat=alat,clat=clat)/2)
200             for i in range(1,len(E)+1):
201                 Ediff.append(E[i-1]-E[len(E)-1])
202             plt.plot(range(2,52,5), E)
203             plt.xlabel('Input Grid Size')
204             plt.ylabel('Energy (eV)')
205             axes = plt.gca()
206             plt.title("Energy")
207             plt.show()
208 
209             plt.plot(range(2, 52, 5), Ediff)
210             plt.xlabel('Input Grid Size')
211             plt.ylabel('Energy Convergence')
212             axes = plt.gca()
213             plt.title("Energy Convergence")
214             plt.show()
215         elif P == 5:
216             E=[]
217             E2=[]
218             alatbcc=2.83
219             alathcp=2.47
220             Vbcc = alatbcc**3
221             ac=1.67
222             Vhcp = ac*0.5*3**(1/2)*alathcp**3
223             for dV in numpy.linspace(1,11,10):
224                 abcc = (Vbcc-dV)**(1/3)
225                 ahcp = (2*(Vhcp-dV)/(ac*3**(1/2)))**(1/3)
226                 chcp=ac*ahcp
227                 E.append(lattice_scan(nk=20,ecut=30,alat=abcc,clat=abcc,anti=True,SM=0.7)/2)
228                 E2.append(lattice_scan(nk=20,ecut=30,alat=ahcp,clat=chcp,anti=True,SM=0.0)
    /2)
229 
230             plt.plot(alatbcc**3-numpy.linspace(1,11,10), E, 'r')
231             plt.plot(ac * 0.5 * 3 ** (1 / 2) * alathcp ** 3 - numpy.linspace(1,11,10), E2, '
    b')
232             plt.xlabel('Unit Cell Volume')
233             plt.ylabel('Energy (eV/atom)')
234             axes = plt.gca()
235             plt.title("Energy")
236             plt.show()
237         elif P == 6:
238             nk=20
239             Eanti=[]
240             Enon=[]
241             Eferro=[]
242             for alat in numpy.linspace(2.5,3.0,10):
243                 Eanti.append(lattice_scan(nk=nk,ecut=30,alat=alat,clat=alat,anti=True,SM=0)
    /2)
244                 Enon.append(lattice_scan(nk=nk, ecut=30, alat=alat, clat=alat, anti=False,SM
    =0.0)/2)
245                 Eferro.append(lattice_scan(nk=nk,ecut=30,alat=alat,clat=alat,anti=False,SM
    =0.7)/2)
246             plt.plot(numpy.linspace(2.5,3.0,10),Eanti,'r')
247             plt.plot(numpy.linspace(2.5,3.0,10),Enon,'b')
248             plt.plot(numpy.linspace(2.5,3.0,10),Eferro,'g')
249             plt.xlabel('Lattice Parameter (A)')
250             plt.ylabel('Energy (eV/atom)')
251             plt.title("Ground State Energy")
252             plt.show()
253 
```

The following function was used to perform the calculations in Problem 2. Like other scripts, the calculations performed are controlled by the Q parameter, though for multiple parts of this question, other parameters must be manually changed, such as the calculation type between scf and relax, or the constraints to allow Ti to move. Q=1 is used for lattice parameter optimization. Q=2 is used (with relaxation) to determine the energy as a function of titanium displacement. Q=3 is used for k-point grid optimization. Q=4 is used for final energy calculation with previously determined lattice parameter, displacement that provides minimal energy, and optimized k-point grid. This last calculation is performed using relaxation and by only constraining the oxygen atoms.

```
    from my_labutil.src.plugins.pwscf import *
from ase.io import write
from ase import Atoms
import matplotlib.pyplot as plt

def make_struc(alat, displacement):
    """
    Creates the crystal structure using ASE.
    :param alat: Lattice parameter in angstrom
    :return: structure object converted from ase
    """
    lattice = alat * numpy.identity(3)
    symbols = ['Pb', 'Ti', 'O', 'O', 'O']
    sc_pos = [[0,0,0], [0.5,0.5,0.5 + displacement], [0,0.5,0.5], [0.5,0,0.5], [0.5,0.5,0]]
    perov = Atoms(symbols=symbols, scaled_positions=sc_pos, cell=lattice)
    # check how your cell looks like
    write('perov.cif', perov)
    structure = Struc(ase2struc(perov))
    return structure



def compute_energy(alat, nk, ecut, disp):
    """
    Make an input template and select potential and structure, and the path where to run
    """
    pseudopots = {'Pb': PseudoPotential(ptype='uspp', element='Pb', functional='LDA', name='
    Pb.pz-d-van.UPF'),
                  'Ti': PseudoPotential(ptype='uspp', element='Ti', functional='LDA', name='
    Ti.pz-sp-van_ak.UPF'),
                  'O': PseudoPotential(ptype='uspp', element='O', functional='LDA', name='O.
    pz-rrkjus.UPF')}
    struc = make_struc(alat=alat, displacement=disp)
    # fix the Pb and Ti atoms in place during relaxation
    constraint = Constraint(atoms={'0': [0, 0, 0]})
    #constraint = Constraint(atoms={'0': [0,0,0], '1': [0,0,0]})
    kpts = Kpoints(gridsize=[nk, nk, nk], option='automatic', offset=True)
    dirname = 'PbTiO3_a_{}_ecut_{}_nk_{}_disp_{}'.format(alat, ecut, nk, disp)
    runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab3/Problem2/relax2", dirname))
    input_params = PWscf_inparam({
        'CONTROL': {
            'calculation': 'relax',
            'pseudo_dir': os.environ['ESPRESSO_PSEUDO'],
            'outdir': runpath.path,
            'tstress': True,
            'tprnfor': True,
            'disk_io': 'none'
        },
        'SYSTEM': {
            'ecutwfc': ecut,
            'ecutrho': ecut * 8,
             },
        'ELECTRONS': {
            'diagonalization': 'david',
            'mixing_beta': 0.7,
            'conv_thr': 1e-7,
        },
        'IONS': {
```

```
            'ion_dynamics': 'bfgs'
        },
        'CELL': {},
        })

    output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
                               params=input_params, kpoints=kpts, constraint=constraint,
    ncpu=1)
    output = parse_qe_pwscf_output(outfile=output_file)
    return output['energy']


def lattice_scan(nk,ecut,alat,disp):
    output = compute_energy(alat=alat, ecut=ecut, nk=nk,disp=disp)
    print(output)
    return output


if __name__ == '__main__':
    # put here the function that you actually want to run
    Q=4

    if Q == 1:
        nk = 4
        ecut = 30
        alat_list = numpy.linspace(3.8, 4.1, 10)
        print(alat_list)
        energy_list = []
        for alat in alat_list:
            energy_list.append(lattice_scan(alat=alat, ecut=ecut, nk=nk, disp=0)/5)
        print(alat_list)
        print(energy_list)
        plt.plot(alat_list, energy_list)
        plt.xlabel('Lattice Parameter (A)')
        plt.ylabel('Energy (eV/atom)')
        axes = plt.gca()
        plt.title("Lattice Parameter Optimization")
        plt.show()

    elif Q == 2:
        nk = 7
        ecut = 30
        alat = 3.8789473684210525
        energy_list=[]
        for disp in numpy.linspace(0.01,0.1,10):
            energy_list.append(lattice_scan(alat=alat,ecut=ecut,nk=nk, disp=disp)/5)
        plt.plot(numpy.linspace(0.01,0.1,10), energy_list)
        plt.xlabel('Relative Displacement')
        plt.ylabel('Energy (eV/atom)')
        axes = plt.gca()
        plt.title("Ti Atom Displacement")
        plt.show()

    elif Q == 3:
        E = []
        Ediff = []
        alat = 3.8789473684210525
        for nk in range(2, 52, 5):
            E.append(lattice_scan(nk=nk, ecut=30, alat=alat,disp=0) / 5)
        for i in range(1, len(E) + 1):
            Ediff.append(E[i - 1] - E[len(E) - 1])
        plt.plot(range(2, 52, 5), E)
        plt.xlabel('Input Grid Size')
        plt.ylabel('Energy (eV/atom)')
        axes = plt.gca()
        plt.title("Energy")
        plt.show()

```

```
        plt.plot(range(2, 52, 5), Ediff)
        plt.xlabel('Input Grid Size')
        plt.ylabel('Energy Convergence (eV/atom)')
        axes = plt.gca()
        plt.title("Energy Convergence")
        plt.show()

    elif Q == 4:
        Energy = lattice_scan(nk=7,ecut=30,alat=3.8789473684210525,disp
    =0.030000000000000006)/5
        print(Energy)

```

The final function was constructed based on the above scripts and previous functions and is used to perform calculations for Problem 3. As in all other cases, a parameter (Q) is used to determine which calculations are performed. For Q=1, lattice parameters are optimized for Cu and Au FCC cells. For Q=2, k-point grid optimization for both cells is performed. Q=3 is used to find the optimal lattice parameters for the CuAu alloy by setting 'calculation' to 'vc-relax' in the input script. Q=4 is used for k-point optimization after finding the optimal parameters for CuAu. Q=5 is used for construction of cells used for visualization in part B, which are later visualized using VESTA. Q=6 performs the energy calculation to determine alloy stability.

```
    from my_labutil.src.plugins.pwscf import *
from ase.spacegroup import crystal
from ase.io import write
from ase.build import *
from ase import Atoms
import numpy
import matplotlib.pyplot as plt

def make_struc(alat,atom,clat):
    """
    Creates the crystal structure using ASE.
    :param alat: Lattice parameter in angstrom
    :return: structure object converted from ase
    """
    if atom == 'Cu' or atom == 'Au':
        fcccell = bulk(atom, 'fcc', a=alat)
        write('fcc.cif', fcccell)
        print(fcccell, fcccell.get_atomic_numbers())
        structure = Struc(ase2struc(fcccell))
    elif atom == 'CuAu':
        lattice = alat * numpy.identity(3)
        lattice[2][2] = clat
        symbols = ['Cu','Au']
        sc_pos = [[0,0,0],[0.5,0.5,0.5]]
        bctcell = Atoms(symbols=symbols, scaled_positions=sc_pos, cell=lattice)
        write('bct.cif', bctcell)
        print(bctcell, bctcell.get_atomic_numbers())
        structure = Struc(ase2struc(bctcell))
    # check how your cell looks like
    print(structure.species)
    return structure

def compute_energy(alat, nk, ecut, atom, clat):
    """
    Make an input template and select potential and structure, and the path where to run
    """

    potpath = os.path.join(os.environ['ESPRESSO_PSEUDO'], 'Cu.pz-d-rrkjus.UPF')
    pseudopots = {'Cu': PseudoPotential(path=potpath, ptype='uspp', element='Cu',
                                        functional='LDA', name='Cu.pz-d-rrkjus.UPF'),
                  'Au': PseudoPotential(path=potpath, ptype='uspp', element='Au',
                                        functional='LDA', name='Au.pz-d-rrkjus.UPF')
                  }
    struc = make_struc(alat=alat, atom=atom, clat=clat)
    if clat != alat:
```

```
46          kpts = Kpoints(gridsize=[nk, nk, int(numpy.floor(nk/2))], option='automatic', offset
    =False)
47      else:
48          kpts = Kpoints(gridsize=[nk, nk, nk], option='automatic', offset=False)
49      dirname = '{}_a_{}_ecut_{}_nk_{}_'.format(atom, alat, ecut, nk)
50      runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab3/Problem3/Ediff", dirname))
51      input_params = PWscf_inparam({
52          'CONTROL': {
53              'calculation': 'scf',
54              'pseudo_dir': os.environ['ESPRESSO_PSEUDO'],
55              'outdir': runpath.path,
56              'tstress': True,
57              'tprnfor': True,
58              'disk_io': 'none',
59          },
60          'SYSTEM': {
61              'ecutwfc': ecut,
62              'ecutrho': ecut * 8,
63              'occupations': 'smearing',
64              'smearing': 'mp',
65              'degauss': 0.02
66               },
67          'ELECTRONS': {
68              'diagonalization': 'david',
69              'mixing_beta': 0.2,
70              'conv_thr': 1e-7,
71          },
72          'IONS': {
73              'ion_dynamics': 'bfgs',
74          },
75          'CELL': {
76              'cell_dynamics': 'bfgs',
77              'cell_factor': 4.0,
78          },
79          })
80
81      output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
82                                 params=input_params, kpoints=kpts, ncpu=1, nkpool=1)
83      output = parse_qe_pwscf_output(outfile=output_file)
84      return output['energy']
85
86  def lattice_scan(nk,ecut,alat,atom,clat):
87      output = compute_energy(alat=alat, ecut=ecut, nk=nk, atom=atom, clat=clat)
88      print(output)
89      return output
90
91  if __name__ == '__main__':
92      # put here the function that you actually want to run
93      Q=6
94
95      if Q == 1:
96          ECu=[]
97          EAu=[]
98          for alat in numpy.linspace(3.9, 4.1, 10):
99              ECu.append(lattice_scan(nk=15,ecut=40,alat=alat,atom='Cu',clat=alat))
100             EAu.append(lattice_scan(nk=15,ecut=40,alat=alat,atom='Au',clat=alat))
101         plt.plot(numpy.linspace(3.4,3.6,10), ECu)
102         plt.xlabel('Lattice Parameter (A)')
103         plt.ylabel('Energy (eV/atom)')
104         axes = plt.gca()
105         plt.title("Cu Lattice Parameter Optimization")
106         plt.show()
107
108         plt.plot(numpy.linspace(3.9, 4.1, 10), EAu)
109         plt.xlabel('Lattice Parameter (A)')
110         plt.ylabel('Energy (eV/atom)')
111         axes = plt.gca()
112         plt.title("Au Lattice Parameter Optimization")
```

```
113         plt.show()
114
115     elif Q == 2:
116         ECu = []
117         EAu = []
118         EdiffCu = []
119         EdiffAu = []
120         alatcu=3.56
121         alatau=4.06
122         for nk in range(5, 45, 5):
123             ECu.append(lattice_scan(nk=nk, ecut=40, alat=alatcu, atom='Cu', clat=alatcu))
124             EAu.append(lattice_scan(nk=nk, ecut=40, alat=alatau, atom='Au', clat=alatau))
125         for i in range(1, len(EAu) + 1):
126             EdiffCu.append(ECu[i - 1] - ECu[len(ECu) - 1])
127             EdiffAu.append(EAu[i - 1] - EAu[len(EAu) - 1])
128         plt.plot(range(5, 45, 5), ECu)
129         plt.xlabel('Input K-Point Grid Dimension')
130         plt.ylabel('Energy (eV/atom)')
131         axes = plt.gca()
132         plt.title("Cu K-Point Grid Optimization")
133         plt.show()
134
135         plt.plot(range(5, 45, 5), EdiffCu)
136         plt.xlabel('Input Grid Size')
137         plt.ylabel('Energy Convergence (eV/atom)')
138         axes = plt.gca()
139         plt.title("Cu Energy Convergence")
140         plt.show()
141
142         plt.plot(range(5, 45, 5), EAu)
143         plt.xlabel('Input K-Point Grid Dimension')
144         plt.ylabel('Energy (eV/atom)')
145         axes = plt.gca()
146         plt.title("Au K-Point Grid Optimization")
147         plt.show()
148
149         plt.plot(range(5, 45, 5), EdiffAu)
150         plt.xlabel('Input Grid Size')
151         plt.ylabel('Energy Convergence (eV/atom)')
152         axes = plt.gca()
153         plt.title("Au Energy Convergence")
154         plt.show()
155
156     elif Q == 3:
157         Energy=lattice_scan(nk=4,ecut=40, alat = 3.81,atom='CuAu',clat=3.80)
158         print(Energy)
159
160     elif Q == 4:
161         ECuAu=[]
162         Ediff=[]
163         alat=2.671769560
164         clat=3.921812470
165         for nk in range(5, 45, 5):
166             ECuAu.append(lattice_scan(nk=nk, ecut=40, alat=alat, atom='CuAu', clat=clat) /
    2)
167         for i in range(1, len(ECuAu) + 1):
168             Ediff.append(ECuAu[i - 1] - ECuAu[len(ECuAu) - 1])
169         plt.plot(range(5,45,5), ECuAu)
170         plt.xlabel('K-Point Input Dimension')
171         plt.ylabel('Energy (eV/atom)')
172         axes = plt.gca()
173         plt.title("CuAu Energy")
174         plt.show()
175
176         plt.plot(range(5, 45, 5), Ediff)
177         plt.xlabel('K-Point Input Dimension')
178         plt.ylabel('Energy Convergence (eV/atom)')
179         axes = plt.gca()
```

```
        plt.title("CuAu Energy Convergence")
        plt.show()

    elif Q == 5:
        #BCT Supercell
        alat=4
        clat=8
        lattice = alat * numpy.identity(3)
        lattice[2][2] = clat
        symbols = ['Cu', 'Au']
        sc_pos = [[0, 0, 0], [0.5, 0.5, 0.5]]
        multiplier = 2*numpy.identity(3)
        multiplier[2][2]=1
        pfctcell = Atoms(symbols=symbols, scaled_positions=sc_pos, cell=lattice)
        pfctcell = make_supercell(pfctcell,multiplier)
        write('pfct.cif', pfctcell)

        #PFCT Cell
        alat2=(2*(alat)**2)**(1/2)
        lattice = alat2 * numpy.identity(3)
        lattice[2][2] = clat
        symbols = ['Cu', 'Cu', 'Au', 'Au']
        sc_pos = [[0, 0, 0], [0.5, 0.5, 0],[0,0.5,0.5],[0.5,0,0.5]]
        pfctcell = Atoms(symbols=symbols, scaled_positions=sc_pos, cell=lattice)
        write('pfct2.cif', pfctcell)

        #Regular BCT Cell
        lattice_scan(4,ecut=40,alat=4,atom='CuAu',clat=8)

    elif Q ==6:
        alatCuAu = 2.671769560
        clatCuAu = 3.921812470
        alatCu = 3.56
        alatAu = 4.06
        nk=15
        ECuAu = lattice_scan(nk=nk,ecut=40,alat=alatCuAu,atom='CuAu',clat=clatCuAu)
        ECu = lattice_scan(nk=nk,ecut=40,alat=alatCu,atom='Cu',clat=alatCu)
        EAu = lattice_scan(nk=nk,ecut=40,alat=alatAu,atom='Au',clat=alatAu)
        Ediff = 0.5*(ECuAu-ECu-EAu)
        print(ECuAu)
        print(ECu)
        print(EAu)
        print(Ediff)

```