Assignment 5 – Computational Materials and Molecular Physics
First hand-in

Eric Lindgren – CID: ericlin

March 6, 2020

1 Task 1: Relax nanoparticles

The computed cohesive energies for the various nanoparticles are given in table 1.

Table 1: Obtained cohesive energies per atom for the various nanoparticles. Note that the cohesive energy increases with the cluster size.

Cluster	Al_{13}	Al_{23}	Al_{38}	Al_{52}	Al_{71}
E_{Coh} (eV)	2.5591	2.7311	2.8374	2.8828	2.9288
Cluster	Al_{90}	Al_{108}	Al_{138}	Al_{201}	Al_{316}
E_{Coh} (eV)	2.9614	2.9865	3.0162	3.0606	3.0999

We note that the cohesive energy per atom increases with cluster size. This is expected, since the cohesive energy is calculated as the average cohesive energy over all atoms in the cluster. The cohesive energy is lower for atoms along the perimiter of the nanoparticle, since they have fewer neighbours. Increasing the size of the nanoparticle thus lowers the impact of these edge-effects, increasing the average cohesive energy.

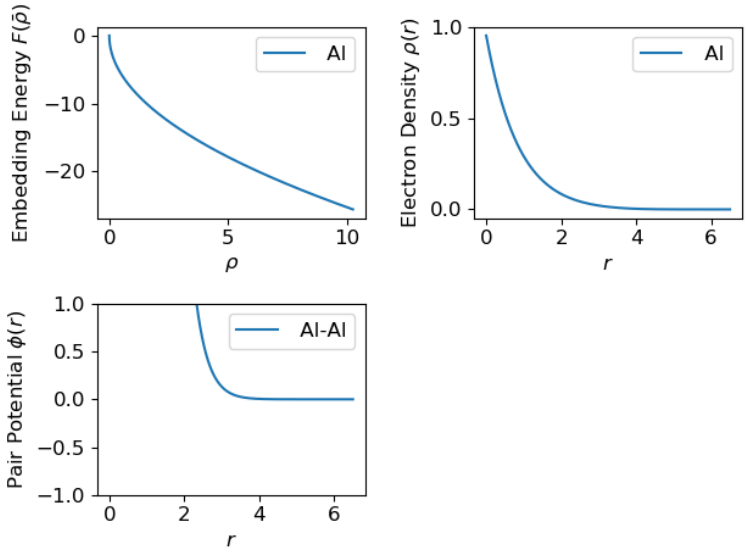

Figure 1: The EAM potential used in task 1. Note that the pair potential seem to have an effective distance of ~ 4 Å, which is around the bond length of Al.

The EAM potential which was used is given in figure 1. Specifically, we note that the pair potential seem to have an effective range of ~ 4 Å, which is around the bond length of crystalline Al (see task 2). Since the bond length is determined by the interatomic forces between all atoms in the crystal lattice, we don't expect the nanoclusters to have the same average interatomic distance as bulk Al, especially not the smaller nanoclusters. Furthermore, the nanoclusters doesn't necessarily exhibit the same axial symmetries as crystalline Al, which also affects the calculations since the potentials used by EAM are equiaxial [1]. Thus, we don't expect the potential to work perfectly for the nanoclusters.

2 Task 2: Obtain bulk Al

A energy-volume relation for bulk Al is given in figure 2. We note that the optimal lattice parameter is around 4.043 Å, with a cohesive energy of $E_{Coh} \sim 3.397$ eV. This cohesive energy is inline with experimental results, where the lattice parameter is $a = 4.0495$ Å and the cohesive energy is $E_{Coh} = 3.39$ eV per atom [2] [3].

We also observe that the cohesive energy is similar to what was obtained in task 1, albeit a bit lower, which is in accordance with the hypothesis that the edge effects decrease with increase particle size. The bulk Al has no edges, due to it's periodic boundary conditions.

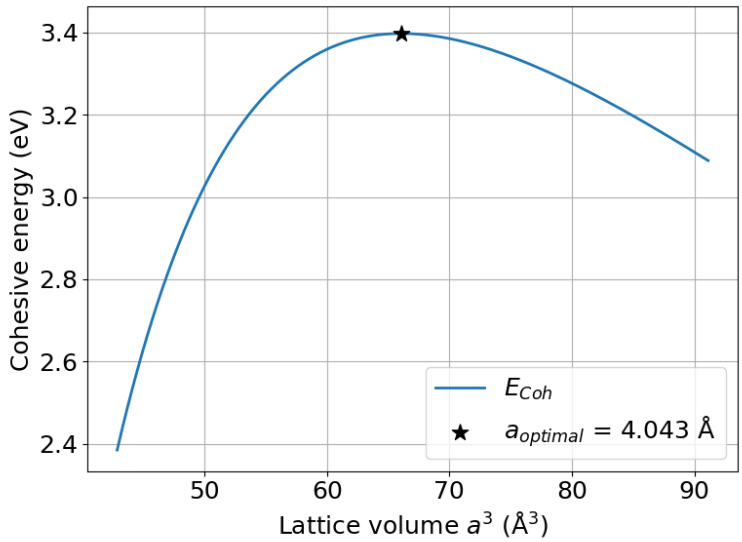

Figure 2: Bulk Al cohesive energy as a function of lattice volume. Note that the maximum of the cohesive energy corresponds to a lattice parameter value of $a \approx 4.043$ Å.

3 Task 3: Vibrational properties of the nanoparticles

The vibrational behaviour for the different nanoparticles were computed using the `Vibrations` module in ASE, see appendix A.3 for implementation details. A summary is given in table 2, with the number of phonon modes (i.e. number of individual resonance frequencies), imaginary frequencies and zero-frequencies. An interesting observation is that the number of zero-frequencies and imaginary frequencies are the same for all systems; upon closer inspection they are the same frequencies; i.e. the frequencies which have zero real part has non-zero imaginary part. The fact that we have some imaginary frequencies imply that the systems are in a transition state [4]; however, the imaginary part is very small ($< 10^{-3}$ cm^{-1}) for all systems except for Al_{13}.

Table 2: Obtained number of modes, imaginary frequnencies and zero-freqncuencies for the various nanoparticles.

Cluster	Vib. modes	Im. freq.	Zero-freq.
Al_{13}	38	4	4
Al_{23}	69	3	3
Al_{38}	114	3	3
Al_{52}	156	3	3
Al_{71}	213	3	3
Al_{90}	270	3	3
Al_{108}	324	3	3
Al_{138}	414	3	3
Al_{201}	574	3	3
Al_{316}	948	3	3

The resulting vibrational density of states are given in figure 3. We observe that the absolute density of states increases with cluster size, which is expected since the larger clusters have more vibrational modes. We note in particular that the vibrational DOS seem to converge to a somewhat bimodal shape for the larger clusters; this will be compared to the results for bulk Al in task 4.

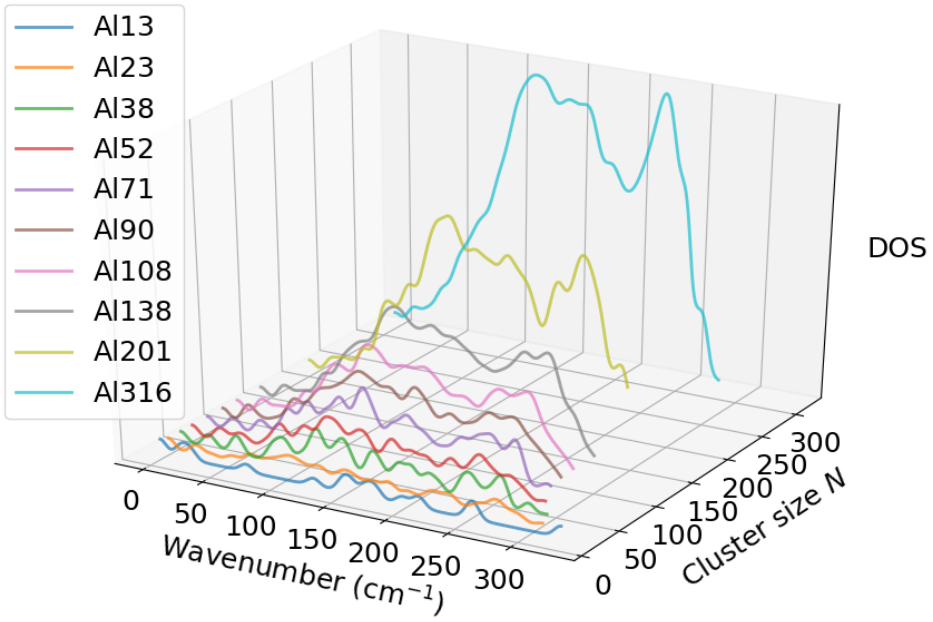

Figure 3: Vibrational DOS for the different nanoclusters. Note that the DOS seems to converge towards a bimodal DOS for larger clusters.

4 Task 4: Vibrational properties of bulk Al

The vibrational DOS as computed by the `Phonons` module in ASE for bulk Al is given in figure 4. See appendix A.4 for implementation details. We note that the vibrational DOS seems to be what the DOS for the individual nanoclusters in task 3 converged towards. This is expected, since the vibrational behaviour of the nanoclusters should become similar to that of bulk Al as the number of atoms increase. In the limit of the number of atoms $N \to \infty$ they should be the same.

The vibrational bandstructure of bulk Al is also given in figure 4. The bandstructure is calculated along the path "GXWKGLUWLK,UX", which passes through many of the high-symmetry points of an FCC crystal [5]. Counting the modes, we determine there to be 3 individual modes, two of which are zero at the Brillouin zone center Γ. The fact that there are modes with zero energy, and thus frequency, at the Brilluin zone center $k = 0$ is that these corresponds to waves with infinite wavelength; thus they corresponds to translating the whole crystal, not deforming it, which costs no energy [6].

We also identify all branches as acoustical, since there is no gap in the frequencies (energies) which the modes describe. The velocity of sound can thus be computed as the slope of the two modes close to the Brillouin zone center (long wavelengths). Each mode corresponds to different translational behaviour, longitudinal and transversal. Generally, the transversal mode has a slower speed of sound than the longitudinal mode [7]. For our Al system, we obtain the sound velocities $\sim 12.0\,\mathrm{km/s}$ and $\sim 17.3\,\mathrm{km/s}$ for the transversal mode and the longitudinal mode, respectively. This is much higher than their experimental values, which should be $3.0\,\mathrm{km/s}$ and $6.4\,\mathrm{km/s}$ respectively. Either the calculation of the band structure is off, or there something wrong with my method, but unfortunately I don't have enough time to investigate this further.

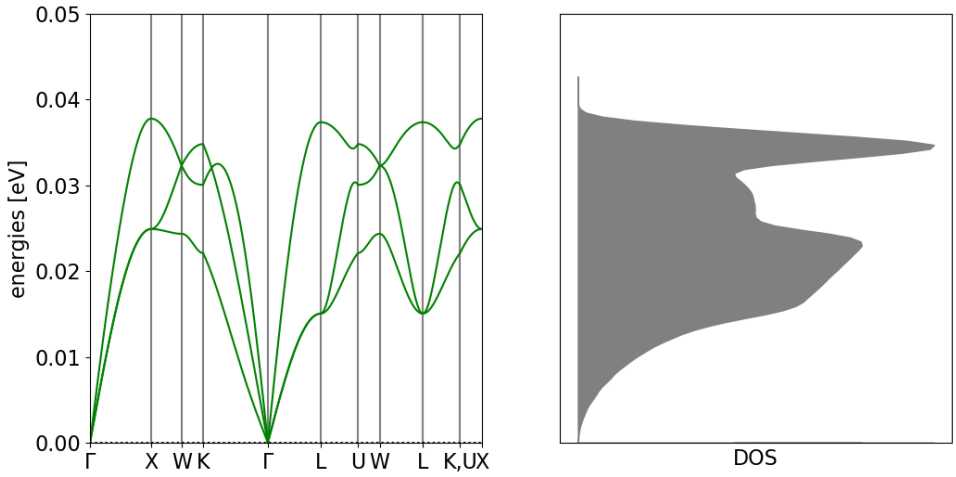

Figure 4: Phonon bandstructure and DOS for bulk Al. Notice that the bandpath has three individual modes, two of which are zero at the Brillouin zone center and that the DOS seems to be what the individual DOS for the nanoparticles converged towards in task 3.

5 Task 5: Electronic structure of nanoparticles

We compute the electronic DOS for the nanoparticles using ASE's `DOS` module and a GPAW calculator. The calculator was set to PW-mode with a cutoff energy of 300 eV in order to compare the results to task 6. The DOS were all folded with Gaussians of width 0.2, and are given in figure 5. As in task 3, the electronic DOS seems to converge towards a distinct profile, which will be compared to what will be obtained in task 6.

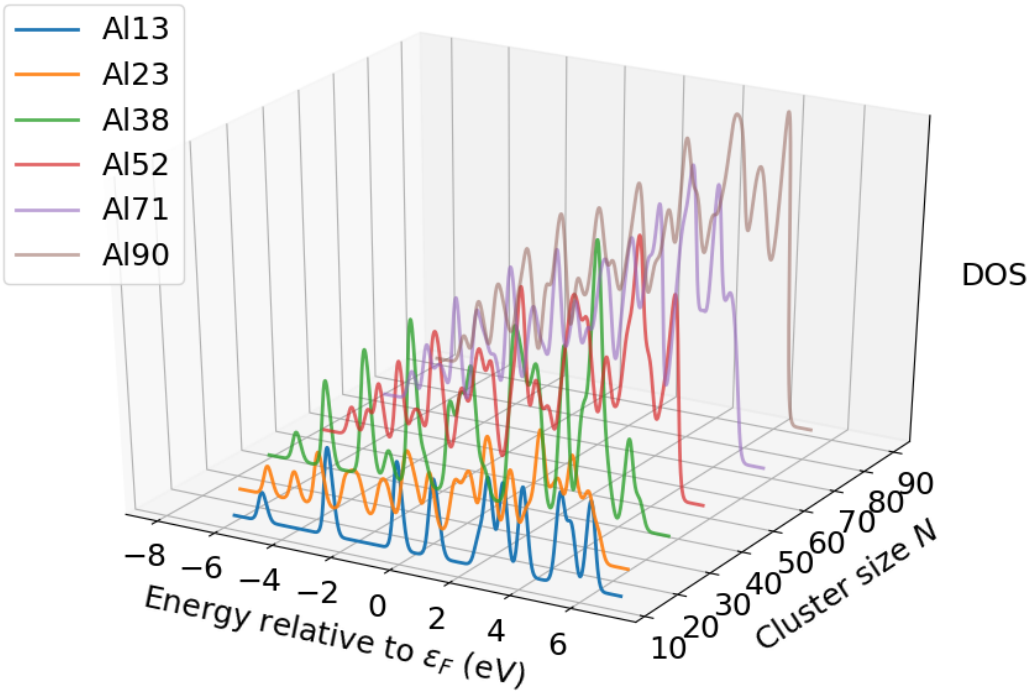

Figure 5: Electronic DOS for the nanoparticles. Note that, as in task 3, the density of states seem to converge to a common shape with increasing cluster size.

6 Task 6: Electronic structure of bulk Al

The electronic band structure and density of states of Al was computed using GPAW, with a plane wave basis and a cutoff energy of 300 eV. To establish a proper sampling of the k-space, the potential energy of the bulk system was converged to 10^{-4} eV by increasing the number of k-points in each direction. A convergence plot of the energy is given in figure 6.

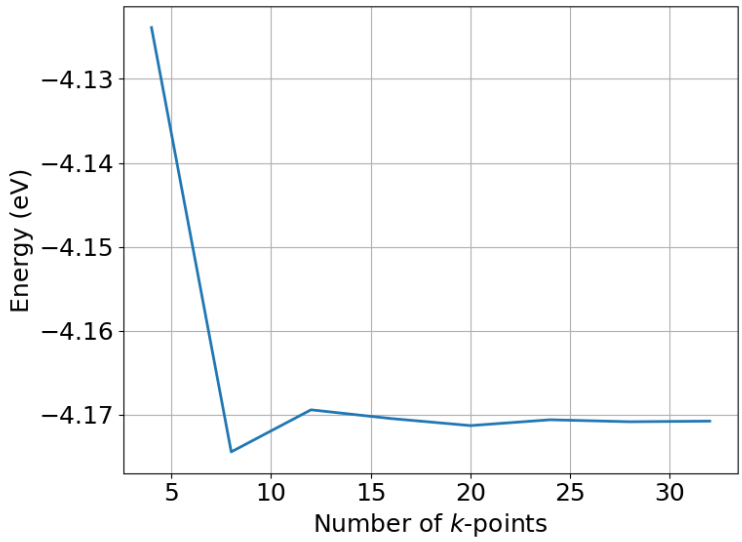

Figure 6: Cluster energy as a function of the number of sampled k-points in each direction.

The ground state potential energy was converged with 36 k-points in each direction. The number of k-points will never be a problem for the nanoparticles, since there we don't have a continuum of states in k-space; we only have a finite number of configurations, and thus we will automatically consider all of them when calculating the DOS.

After this the electronic ground state energy was calculated again and fixated. This fixated density was then used to calculate the electronic band structure of bulk Al. The band structure was calculated along the same path as in task 4, "GXWKGLUWLK,UX", with 60 sampled k-points and convoluted with a Gaussian of width 0.2. Finally, the electronic DOS was calculated, but not along a path but on a mesh on K-space with 40 k-points in each direction in order to properly sample the k-space. The DOS was convoluted with Gaussians of width 0.2. The resulting electronic band structure and DOS are given in figure 7. Note that both the band structure and the DOS are plotted relative to the Fermi level ϵ_F. We observe that the DOS for bulk Al is somewhat similar to the converged spectrum in task 5. However, there are some discrepancies, such as the large peaks at $\sim -2.5\,\text{eV}$ in figure 7, which are non-existent in figure 5. This could be due to me using different methods for calculating the DOS in task 5 and task 6, and it is possible that I have configured them improperly so that they don't converge to the same degree. I ran into calculation optimization problems when trying to use the same procedure for task 5 as I used for task 6.

We also compare the DOS for bulk Al with what is predicted by the free electron model. The DOS for metals in the free electron model is given as proportional to the square-root of the energy, $g_{Free} \propto \sqrt{E}$ [8]. By rescaling the free-electron DOS to fit the obtained DOS we see that it describes the location of the first few peaks fairly well. We don't expect perfect one-to-one correspondence with the free electron model, since it models the electrons as having a continuum of accessible states, whilst the DFT approach does no such approximation.

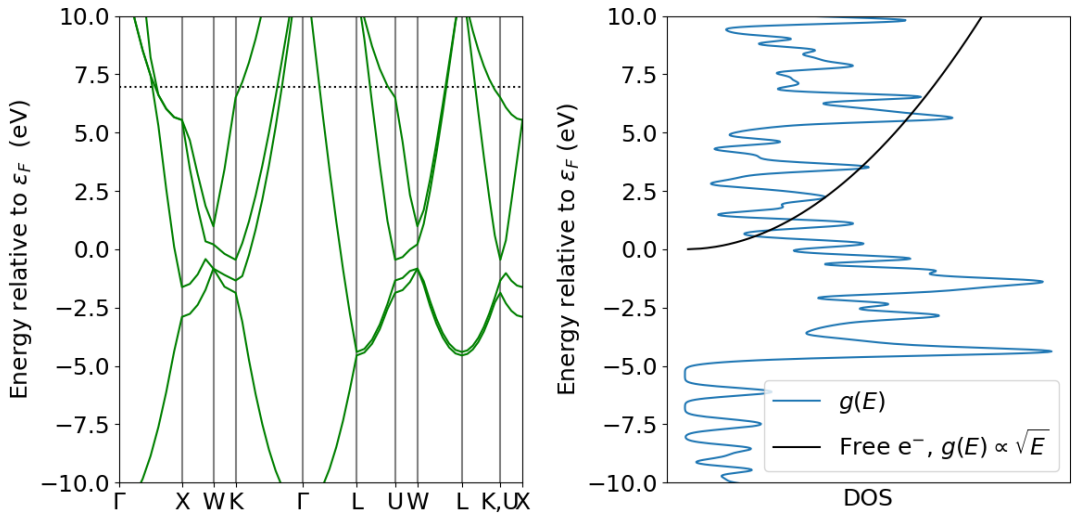

Figure 7: Electronic band structure and DOS for bulk Al. Disregard the horizontal line in the band spectrum plot; it is an artefact from plotting.

7 Task 7: Electrons and phonons in Si

The electronic and vibrational properties of Si was calculated using the same methods as in tasks 4 and 6, but adapted for Si. The electronic structure was calculated with GPAW and a plane wave basis, but with a cutoff of 200 eV. Here the ground state energy was converged to 10^{-4} eV using only 16 k-points. The band structure was evaluated along the path "GXWKL" sampled at 60 points, and the DOS was calculated on a mesh spanning 28 k-points in each direction.

The electronic bandgap was calculated using the `Bandgap` module in ASE. Comparing the bandgap energy with the module set to calculate either the direct or indirect bandgap, we find that the bandgap is the smallest for the indirect bandgap, with a value of $E_{gap} = 0.475$ eV. Thus we draw the conclusion that bulk Si has an indirect bandgap. Comparing the bandgap to the experimental value of 1.14 eV we also note that the DFT calculation seems to underestimate the bandgap value [9]. This is probably due to me having used LDA as my XC-functional in order to simplify the calculations; LDA corresponds to local DFT, which often tends to underestimate the band gap [10].

Unfortunately, I didn't have time to calculate the effective electron and hole masses, but it can be done by fitting a second degree polynomial to the bandstructure at the relevant positions in the Brillouin zone and extracting the second derivative of the curve f''. The effective mass can then be calculated as $m* = f''/\hbar^2$ [11].

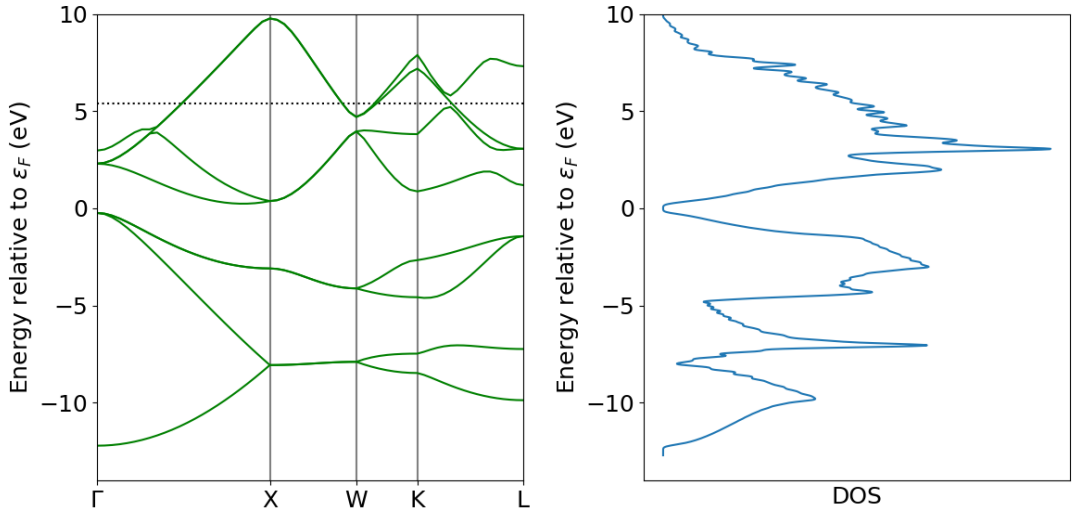

Figure 8: Electronic structure for Si. Disregard the horizontal line in the band spectrum plot; it is an artefact from plotting. Notice the bandgap which is clearly visible in the DOS.

The vibrational structure and DOS was also calculated using DFT and a GPAW plane wave calculator, contrary to what was done in task 4. However, the `Phonons` module was also used here. We identify 6 different phonon modes in total; three acoustical and three optical. The optical branches are the ones which never goes to zero, i.e. they only occupy a range of frequencies with their lowest frequency greater than zero. Thus we may get a phonon frequency gap, depending on our position in the Brillouin zone. We also note that among our acoustic branches only two are zero at the Brillouin zone center Γ. By studying these two acoustic branches' behaviour close to Γ we may get an estimate for the speed of sound in Si, as we got for Al in task 4. The obtained value for the speed of sound in Si for the transversal (slower) branch is $\sim 2500\,\mathrm{km/s}$ and for the longitudinal branch $\sim 19\,\mathrm{km/s}$. This is, as in task 4, very different from the experimental values of $5.5\,\mathrm{km/s}$ and $8.4\,\mathrm{km/s}$ respectively [7], and like in task 4 I unfortunately don't have the time to investigate this further.

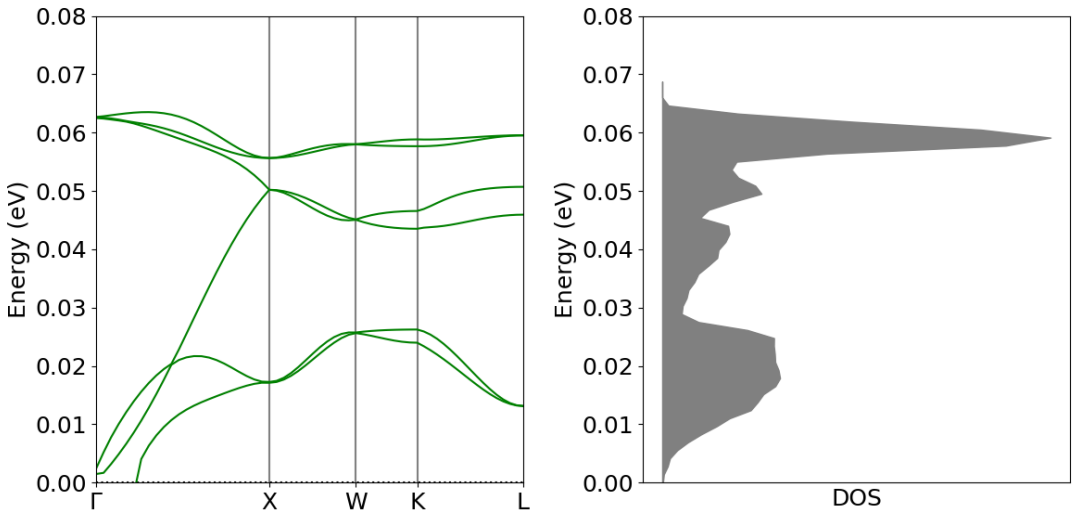

Figure 9: Vibrational structure for Si. Notice the division into three optical and three acoustic phonon modes.

References

[1] ASE-developers. *EAM – ASE documentation*. Accessed: 2020-03-06. 2017. URL: %5Curl%7Bhttps://wiki.fysik.dtu.dk/ase/ase/calculators/eam.html%7D.

[2] periodictable.com. *Lattice Constants of the elements*. Accessed: 2020-03-06. 2020. URL: %5Curl%7Bhttps://periodictable.com/Properties/A/LatticeConstants.html%7D.

[3] Knowledgedoor. *Cohesive energy*. Accessed: 2020-03-06. 2005. URL: %5Curl%7Bhttp://www.knowledgedoor.com/2/elements_handbook/cohesive_energy.html%7D.

[4] University of Waterloo. *Transition states*. Accessed: 2020-03-06. URL: %5Curl%7Bhttp://scienide2.uwaterloo.ca/~nooijen/Chem-440-computational/Lab_Gaussian_transition_states.pdf%7D.

[5] ASE-developers. *Brillouin zone sampling*. Accessed: 2020-03-06. 2017. URL: %5Curl%7Bhttps://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.html#ase.dft.band_structure.BandStructure%7D.

[6] Wikipedia. *Phonon*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Phonon%7D.

[7] Wikipedia. *Speeds of sound of the elements*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Speeds_of_sound_of_the_elements%7D.

[8] Wikipedia. *Free electron model*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Free_electron_model%7D.

[9] Hyperphysics. *Semiconductor Band gaps*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttp://hyperphysics.phy-astr.gsu.edu/hbase/Tables/Semgap.html%7D.

[10] Materials Project. *Al*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttps://materialsproject.org/materials/mp-134/?__cf_chl_jschl_tk__=abb4d39ddf5a397b50378ddbc98d5327cce7e49f-1583469421-0-AZqFiCLZ4h2Ct1E1KvRvHHDsEzFPMO0tJbyqTBvR8hzJ3FRYUGLI54OyugUxd9hhGSrMhSlnAzywPhEMACS94UwQ5pZu5z7uLAhQIaNmP9S2_mRhotUGqw2qqYqX5S9jkgX3xHIOfsHTL5wQIEUnAQU_5yqFHMpA_U3zLGbIVQ2U4X03DXEP-WoKSBzkkd1oVOm1OSYcoc9-ekxTLD-hDtxDqEd4Df5EBR2ksIT4ULG7D.

[11] Wikipedia. *Effective mass (solid-state physics)*. Accessed: 2020-03-06. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Effective_mass_(solid-state_physics)%7D.

A Python scripts

A.1 Task 1

Please run both scripts in the folder ”task1”.

```python
# External imports
import numpy as np
import matplotlib.pyplot as plt

# ASE
from ase import Atoms
from ase.db import connect
from ase.calculators.eam import EAM


# Set plot params
plt.rc('font', size=12)          # controls default text sizes
plt.rc('axes', titlesize=12)     # fontsize of the axes title
plt.rc('axes', labelsize=12)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=12)    # fontsize of the tick labels
plt.rc('ytick', labelsize=12)    # fontsize of the tick labels
plt.rc('legend', fontsize=12)    # legend fontsize

def out(s, f):
    ''' Prints s to console and file f. '''
    print(s)
    print(s, file=f)

# Load the database, row by row, and calculate the cohesive energy.
# The cohesive energy is the same as the potential energy since that is
# the energy required to separate the atoms.
db = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
# Sort the clusters based on number of atoms
allClust = list(db.select())
sort = np.argsort([len(clust.numbers) for clust in allClust])
allClust = np.array(allClust)[sort]

Efile = open('ECoh.txt', 'w')
out(s='-----------    Cluster energies    ----------', f=Efile)
for clust in allClust:
```

```
    # General info
    atoms = clust.toatoms()
    N = len(atoms.positions)
    # Calculate cohesive energy
    mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
    atoms.set_calculator(mishin)
    ECoh = np.abs(atoms.get_potential_energy() / N)  # Cohesive energy is the ↩
        positive potential energy

    # Print results
    str1 = f'| N={N}'
    str2 = 'Cohesive energy/atom:'
    str3 = f' {ECoh:.4f} eV   |'
    out(str1 + str2.rjust(30-len(str1)) +  str3.rjust(35-len(str2)), f=Efile)
out('-------------------------------------------', f=Efile)

# Plot and save a picture of the potential
mishin.plot()
plt.tight_layout()
plt.savefig('al_potential.png')
# plt.show()

# The potential may not be perfectly suited for the smaller nanoparticles, since they↩
    don't exhibit the same symmetries as the bulk Al.
# Bulk Al is crystalline fcc.
```

A.2 Task 2

Please run in the folder ”task2”.

```
# External imports
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect
from ase.calculators.eam import EAM
from ase.build import bulk


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize


# The true lattice parameter is 4.04    - search around this minimum
latParams = np.linspace(3.5, 4.5, 1000)

E = []
for a in tqdm(latParams):
    atoms = bulk('Al', 'fcc', a)

```

```
    # Calculate potential energy
    mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
    atoms.set_calculator(mishin)
    E.append(atoms.get_potential_energy())

E = np.abs(E)  # Cohesive energy is abs of potential energy
# Find maxmum cohesive energy and best lattice parameter
maxIdx = np.argmax(E)
aMax = latParams[maxIdx]
EMax = E[maxIdx]
print(f'Optimal lattice parameter: {aMax:.3f}   , Cohesive energy: {EMax:.3f} eV')

# Plot results
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(latParams**3, E, linewidth=2, linestyle='-', marker='o', markersize=0, color='C0', label=r'$E_{Coh}$')
ax.scatter(aMax**3, EMax, marker='*', s=150, c='k',zorder=3, label=r'$a_{optimal}$ ' + f'= {aMax:.3f}   ')
ax.set_xlabel(r'Lattice volume $a^3$ ($\rm ^3$)')
ax.set_ylabel(r'Cohesive energy (eV)')
ax.legend(loc='best')
ax.grid()
plt.tight_layout()
plt.savefig('E_task2.png')
# plt.show()

# The lattaice parameter agrees with experimental values up to 2 decimal
# places, and the cohesive energy seems to match those (per atom)
# from task1 somewhat well.
# However, the cohesive energy for bulk aluminium averaged over all atoms
# is lower, which could be due to the increased stability of the bulk crystal.
# In the nanoparticles there are atoms on the surface which are bound less
# tight which reduces the average cohesive energy. The effects of these
# ''surface atoms'' decreases with the size of the nanoparticle which can be
# seen by the cohesive energy increasing with the number of atoms in the
# nanoparticle. The bulk Al corresponds to the cohesive energy of an infinite
# size nanoparticle, i.e. no edge effects.

# We expect EAM to give good results, since it is particularly good for FCC metals
# [https://wiki.fysik.dtu.dk/ase/ase/calculators/eam.html].
```

A.3 Task 3

Please run in the folder ”task3”.

```
# Internal imports
import os.path as p

# External imports
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect
from ase.calculators.eam import EAM
from ase.vibrations import Vibrations
```

```
from ase.phonons import Phonons


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

def out(s, f):
    ''' Prints s to console and file f. '''
    print(s)
    print(s, file=f)

# Load the database, row by row, and calculate the cohesive energy.
# The cohesive energy is the same as the potential energy since that is
# the energy required to separate the atoms.
db = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
vibDB = connect('./vib.db', append=False)
# Sort the clusters based on number of atoms
allClust = list(db.select())
sort = np.argsort([len(clust.numbers) for clust in allClust])
allClust = np.array(allClust)[sort]

for clust in allClust:
    # General info
    atoms = clust.toatoms()
    N = len(atoms.positions)
    # Define calculator - mishin
    mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
    atoms.set_calculator(mishin)
    # Get vibrational spectrum
    str1 = f'--------        N={N}'
    str2 = '        ---------'
    print(str1 +  str2.ljust(40-len(str1)))

    ##### Using Vibrations module
    v = Vibrations(atoms, name=f'./vibs/vib_{N}')
    if not p.isfile(f'./vibs/vib_{N}.all.pckl'):
        print('Running vibration calculation')
        v.run()
        v.combine()  # Combine pickle files
    # Get frequencies and DOS - i.e # of states per frequency
    all_freq = v.get_frequencies()
    # if N==38:
    #     print(v.summary())
    #     print(all_freq)
    (freq, counts) = np.unique(all_freq, return_counts=True)
    fold_freq = v.fold(np.real(freq), np.real(counts), start=0, end=np.real(freq.max↩
        ()), width=12, normalize=False)
    f_freq = np.array(fold_freq[0])
    f_dos = np.array(fold_freq[1])
    freq = np.array(freq)
    dos = np.array(counts)

    # Get number of modes
    # e = False
    # i=0
    # modes = 0
    # while not e:
```

```
    #     try:
    #         v.get_mode(i)
    #         modes += 1
    #         i += 1
    #     except:
    #         e = True

    # Save to db
    vibDB.write(atoms, data={'frequency': freq, 'DOS': dos, 'f_freq': f_freq, 'f_dos': f_dos})

    ##### Using Phonons module
    # ph = Phonons(atoms, mishin, delta=0.05, name='./phonons/ph_Si')
    # ph.run()
    # # ph.combine()
    # ph.read(acoustic=True)
    # Pdos = ph.get_dos(kpts=(20, 20, 20)).sample_grid(npts=60, width=1e-3)
    # print(Pdos)

print('Sucessfully saved all data to DB')
```

```
# External imports
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

def out(s, f):
    ''' Prints s to console and file f. '''
    print(s)
    print(s, file=f)

# Read DB
vibDB = connect('./vib.db')

# Plot vibrational spectra
vFile = open('vib.txt', 'w')
out(s='-----------------------------    Cluster Vibrations    -----------------------------', f=vFile)
fig = plt.figure(figsize=(9,6))
ax = fig.add_subplot(111, projection='3d')
for clust in vibDB.select():
    atoms = clust.toatoms()
    freq = clust.data['frequency']
    dos = clust.data['DOS']
    # Extract info
    N = len(atoms.positions)
```

```
    nModes = len(freq)
    isC = np.iscomplex(freq).sum()
    isZ = len(freq) - np.count_nonzero(np.real(freq))
    # Print info
    str1 = f'| Al{N} '
    str2 = 'Modes: '
    str3 = f' {nModes} '
    str4 = '    Imaginary frequencies: '
    str5 = f' {isC} '
    str6 = '    Zero frequencies: '
    str7 = f' {isZ} |'
    out(str1 + str2.rjust(20-len(str1)) +  str3.rjust(15-len(str2)) + str4 + str5 + str6 + str7, f=vFile)
    # Plot
    f_freq = clust.data['f_freq']
    f_dos = clust.data['f_dos']
    f_freq = np.real(f_freq)
    ax.plot(f_freq, N*np.ones(len(f_freq)), f_dos, alpha=0.7, linewidth=2, label=f'Al{N}')
ax.grid()
ax.legend(loc='upper left')
ax.set_xlabel(r'Wavenumber ($\rm cm^{-1}$)', labelpad=15)
ax.set_ylabel(r'Cluster size $N$', labelpad=15)
# ax.set_zlabel(r'DOS ($\rm cm$)', labelpad=10) # TODO set proper units
ax.set_zlabel(r'DOS')
ax.set_zticks([])
ax.grid()
plt.tight_layout()
plt.savefig('figTask3.png')
out(s='
    ----------------------------------------------------------------------------------------
    ', f=vFile)
```

A.4 Task 4

Please run in the folder ”task4”.

```
# Internal imports
import os.path as p

# External imports
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect
from ase.calculators.eam import EAM
from ase.build import bulk
from ase.vibrations import Vibrations
from ase.phonons import Phonons
from ase.visualize import view


# Set plot params
plt.rc('font', size=16)          # controls default text sizes
plt.rc('axes', titlesize=16)     # fontsize of the axes title
```

```
plt.rc('axes', labelsize=16)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=16)    # fontsize of the tick labels
plt.rc('ytick', labelsize=16)    # fontsize of the tick labels
plt.rc('legend', fontsize=16)    # legend fontsize

def soundV(bs, modes):
    # Get K-points and energies
    k = bs.path.kpts
    e = bs.energies
    N = 20
    v = 0
    hbar = 6.582e-16 # eVs
    fig, ax = plt.subplots(figsize=(8,6))
    for i in range(modes):
        omegar = e[0,:N,i] / hbar
        kr = k[:N,i]
        if not kr[-1] == 0:
            # The last mode is overlayed on one of the others
            v = np.abs( (omegar[-1] - omegar[0]) / (kr[-1] - kr[0]) ) / 1e10 #   /s
            print(v)
            ax.plot(k[:N,i], e[0,:N,i])
    return None

# DBs
vibDB = connect('./vib.db', append=False)  # DB for vibration spectrum
phonDB = connect('./ph.db', append=False)  # DB for phonon structure and DOS
# Using optimal lattice parameter from task 2
a = 4.043  # A
atoms = bulk('Al', 'fcc', a)
# view(atoms)  # DEBUG

#### Calculate vibrational spectrum
# Attach EAM calculator to atoms
mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
atoms.set_calculator(mishin)

# Get vibrational spectrum
v = Vibrations(atoms, name='./vibs/vib_bulk')
v.run()
v.summary()
# Get frequencies and DOS - i.e # of states per frequency
(freq, counts) = np.unique(v.get_frequencies(), return_counts=True)
freq = np.array(freq)
dos = np.array(counts)
# Save to db
vibDB.write(atoms, data={'frequency': freq, 'DOS': dos})

#### Calculate band structure
N=7 # Use a supercell of size 7x7x7
ph = Phonons(atoms, mishin, name='./phonons/ph_bulk', supercell=(N,N,N), delta=0.05)
ph.run()
# Read the results from the run and obtain the bandpath and DOS
ph.read(acoustic=True)
# ph.clean()

# lat.plot_bz(show=True) # Visualize Brillouin zone
# Al has Space Group 225 [https://materialsproject.org/materials/mp-134/]
# from which Bilbao Cryst gives us
# Also here is given optimal vectors https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.↩
    html
# Use default path for now
# And here [https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.html#ase.dft.band_structure↩
```

```
    .BandStructure]
path = atoms.cell.bandpath(path='GXWKGLUWLK,UX', density=100)

bs = ph.get_band_structure(path)
modes = bs.energies.shape[2]
print(f'Number of phonon modes: {modes}')

# Compute sound velocity
v = soundV(bs, modes)

dos = ph.get_dos(kpts=(20,20,20)).sample_grid(npts=100, width=1e-3)

# Plot phonon spectrum and DOS
fig, ax = plt.subplots(1,2, figsize=(12,6))
emax = 0.05
bs.plot(ax=ax[0], emin=0.0, emax=emax)

# ax[1] = fig.add_axes([.8, .07, .17, .85])
ax[1].fill_between(dos.weights[0], dos.energy, y2=0, color='grey',
                   edgecolor='k', lw=1)

ax[1].set_ylim(0, emax)
ax[1].set_yticks([])
ax[1].set_xticks([])
# ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
ax[1].set_xlabel(r'DOS') # TODO set proper units

plt.savefig('phonon_task4.png')
plt.tight_layout()
```

A.5 Task 5

Please run in the folder ”task5”.

```
# Internal imports
import time
import pickle

# External imports
import numpy as np

# ASE
from ase import Atoms
from ase.db import connect
from ase.dft.dos import DOS
from ase.parallel import world

# GPAW
from gpaw import GPAW, PW

'''
Perform GPAW DFT calculation of the electron density of states
for the nanoparticles with N < 100.
'''
# Connect to DB
structDB = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
eosDB = connect('./eos.db', append=False)

```

```python
# Sort the clusters based on number of atoms
allClust = list(structDB.select())
sort = np.argsort([len(clust.numbers) for clust in allClust])
allClust = np.array(allClust)[sort]

for clust in allClust:
    # General info
    atoms = clust.toatoms()
    N = len(atoms.positions)
    if(N<100):
        start = time.time()
        # if world.rank == 0:
        print(f'Calculating EOS for Al{N}')

        # Define electron calculator (GPAW)
        calc = GPAW(
            mode=PW(300),  # Lower for computational efficiency
            txt=f'./gpaw-out/EOS_{N}_1core.txt'
        )  # Use the same calculator as in task6
        atoms.set_calculator(calc)
        pot_e = atoms.get_potential_energy()  # Self-constistently optimize the electron density
        # if world.rank == 0:
        print(f'Cluster Al{N} finished potential energy per atom: {pot_e / N:.2f} eV')

        # Get the electronic DOS
        dos = DOS(calc, npts=800, width=0.2)

        e = dos.get_energies()
        d = dos.get_dos()
        e_f = calc.get_fermi_level()
        e -= e_f  # Subtract the Fermi level from the energy

        ##### Get the DOS using the same method as in task6
        # print('Electronic band structure calculated')
        # kpts = {'size': (40,40,40)}
        # calc.set(
        #     kpts = kpts,
        #     fixdensity=True,
        #     symmetry='off',
        # )
        # # Fix the potential
        # calc.get_potential_energy()
        # e, dos = calc.get_dos(spin=0, npts=1001, width=0.5)  # Get energy and density of states
        # e_f = calc.get_fermi_level()

        # Edos = {
        #     'e': e,
        #     'dos': dos,
        #     'fermi': e_f
        # }

        # # Save results
        # pickle.dump( Edos, open( f'./dos/Edos_Al{N}_1core.p', "wb" ) )  # Save the electronic DOS

        end = time.time()
        # if world.rank == 0:
        print(f'Cluster Al{N} finished ---- Time: {(end-start):.2f} s')
        eosDB.write(atoms, data={'energy': e, 'DOS': d, 'fermi': e_f})
```

```
        calc.write(f'./calculators/calc{N}.gpw')  # Save the calculator
    else:
        # if world.rank == 0:
        print(f'Skipping Al{N}')
```

```
# External imports
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

# Connect to DB
eosDB = connect('./eos.db')

# Extract and plot convergence data
# fig, ax = plt.subplots(figsize=(8,6))

fig = plt.figure(figsize=(9,6))
ax = fig.add_subplot(111, projection='3d')
dbList = list(eosDB.select())

for i,row in enumerate(dbList):
    atoms = row.toatoms()
    N = len(atoms.positions)
    dos = row.data['DOS']
    e = row.data['energy'] - row.data['fermi']
    ax.plot(e, N*np.ones(len(e)), dos, alpha=1-i*0.1, linewidth=2, label=f'Al{N}')

ax.legend(loc='upper left')
ax.set_xlabel(r'Energy relative to $\epsilon_F$ (eV)', labelpad=15)
ax.set_ylabel(r'Cluster size $N$', labelpad=15)
# ax.set_zlabel(r'DOS ($\rm eV^{-1}$)', labelpad=10) # TODO set proper units
ax.set_zlabel(r'DOS')
ax.set_zticks([])
ax.grid()
plt.tight_layout()
plt.savefig('dos_task5')
plt.show()
```

A.6 Task 6

Please run in the folder ”task6”.

```python
# Internal imports
import time
import pickle

# External imports
import numpy as np

# ASE
from ase import Atoms
from ase.db import connect
from ase.build import bulk
from ase.parallel import world

# GPAW
from gpaw import GPAW, PW, restart

'''
Perform GPAW DFT calculation for the electronic structure of
bulk Al. First find converge number of k-points. Then, converge
density self-consistently.
'''

def convergeK(atoms, tol=1e-4, kstart=4):
    # Converge total energy by increasing k-space sampling until total energy changes by
    # <10^-4 eV.

    # DBs
    convDB = connect('./bulk.db', append=False)  # DB for electronic spectrum

    k = kstart
    Etot_old = 1
    Etot_new = 2
    E = []
    ks = []
    i = 1
    while np.abs(Etot_new - Etot_old) > tol:
        start = time.time()
        Etot_old = Etot_new
        # if world.rank == 0:
        print(f'---- Iteration: {i} ---- k={k} ----')

        calc = GPAW(
                mode=PW(300),                 # cutoff
                kpts=(k, k, k),               # k-points
                txt=f'./gpaw-out/k={k}.txt'   # output file
            )
        atoms.set_calculator(calc)
        Etot_new = atoms.get_potential_energy()  # Calculates the total DFT energy of the bulk material
        end = time.time()

        # if world.rank == 0:
        print(f'Energy: {Etot_new:.4f} eV ---- Time: {(end-start):.2f} s')
        E.append(Etot_new)
        ks.append(k)
        k += 4
        i += 1
    # Save calculator state and write to DB
    # if world.rank == 0:
    convDB.write(atoms, data={'energies': E, 'ks': ks})
    calc.write('kConverge.gpw')
```

```
    print('Written to DB')
    return k, calc


# Using optimal lattice parameter from task 2
a = 4.043  # A
atoms = bulk('Al', 'fcc', a)

# Find optimal k parameter
k, calc = convergeK(atoms, tol=1e-4, kstart=4)
print(f'Optimal k-parameter: k={k}')

# Perform a ground state energy calculation to get the ground state density
atoms.get_potential_energy()

# Save the calculator
calc.write('Al_calc.gpw')
# if world.rank == 0:
print('Calculator saved')

#### Electronic band structure
# if world.rank == 0:
print('Electronic structure calculation started')
atoms, calc = restart('Al_calc.gpw')
# kpts = {'size': (60,60,60), 'path': 'GXWKGLUWLK,UX'}
kpts = {'path': 'GXWKGLUWLK,UX', 'npoints': 60}
calc.set(kpts = kpts, fixdensity=True, symmetry='off')

# calc = GPAW(
#     'Al_calc.gpw',
#     nbands=16,                              # Include more bands than convergence since metallic
#     fixdensity=True,                        # Fixate the density
#     symmetry='off',                         # Check all points along the path
#     kpts={'path': 'GXWKL', 'npoints': 60},
#     convergence={'bands': 8},
#     txt='Al_calc.txt'
# )
# calc.get_potential_energy()  # Converge the system
# # if world.rank == 0:
# print('Electronic structure converged')

# Fix the potential
calc.get_potential_energy()

# Get band structure and dos
Ebs = atoms.calc.band_structure()  # Get the band structure

# if world.rank == 0:
print('Electronic band structure calculated')
kpts = {'size': (40,40,40)}
calc.set(
    kpts = kpts,
    fixdensity=True,
    symmetry='off',
)
# Fix the potential
calc.get_potential_energy()

e, dos = calc.get_dos(spin=0, npts=1001, width=0.5)  # Get energy and density of states
print('Electronic DOS computed')
```

```
e_f = calc.get_fermi_level()
Edos = {
    'e': e,
    'dos': dos,
    'fermi': e_f
}

# Save results
pickle.dump( Ebs, open( "Ebs.p", "wb" ) )  # Save the electronic band structure
pickle.dump( Edos, open( "Edos.p", "wb" ) )  # Save the electronic DOS
# if world.rank == 0:
print('Electronic structure calculation completed')
```

```
# Internal imports
import pickle

# External imports
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

# Connect to DB
bulkDB = connect('./bulk.db')

# Extract and plot convergence data
fig, ax = plt.subplots(figsize=(8,6))
dbList = list(bulkDB.select())

ks = dbList[0].data['ks']
E = dbList[0].data['energies']
ax.plot(ks, E, linewidth=2)

ax.set_xlabel(r'Number of $k$-points')
ax.set_ylabel('Energy (eV)')
ax.grid()
plt.tight_layout()
plt.savefig('kConvergence.png')

# Extract and plot band electronic band structure and DOS
bs = pickle.load(open( "Ebs.p", "rb" ))
d = pickle.load(open( "Edos.p", "rb" ))

fig, ax = plt.subplots(1,2, figsize=(12,6))

# BS
bs.energies = bs.energies - d['fermi']
emax = 10
```

```
emin = -emax
bs.plot(filename='', ax=ax[0], show=False, emax=emax, emin=emin)
ax[0].set_ylabel(r'Energy (eV)')

# DOS
e = d['e']-d['fermi']
# ax[1].fill_between( d['dos'], e, y2=0, color='grey',
#                     edgecolor='k', lw=1, alpha=0.6, label=r'$g(E)$')
ax[1].plot(d['dos'], e, label=r'$g(E)$' )

ePos = np.array([ ei for ei in e if ei>=0 ])
# Calculate free electron density
# Na = 6.02214076e23  # Avogadro's constant
# Z = 3 # Nbr of valence electrons of Al
# rho = 2720 # kg/m3
# ma = 26.98 * 1.66e-27
# n = Na*Z*rho/ma
# freeE = 1.5 * n/d['fermi'] * np.sqrt(ePos/d['fermi'])
freeE = 0.115*np.sqrt(ePos)  # TODO set proper scale
ax[1].plot(freeE, ePos, color='k', label=r'Free $\rm e^{-}$, $g(E) \propto \sqrt{E}$')

ax[1].legend(loc='best')
ax[1].set_xticks([])
# ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
ax[1].set_xlabel(r'DOS')
ax[1].set_ylabel(r'Energy relative to $\epsilon_F$ (eV)')
ax[1].set_ylim((emin, emax))
plt.tight_layout()
plt.savefig('electronicAl.png')
```

A.7 Task 7

Please run in the folder ”task7”.

```
# Internal imports
import time
import pickle

# External imports
import numpy as np

# ASE
from ase import Atoms
from ase.db import connect
from ase.build import bulk
from ase.phonons import Phonons
from ase.parallel import world

# GPAW
from gpaw import GPAW, PW, restart

'''
Perform DFT calculation for electronic and phononic band structures and density of states for Si.
Uses a GPAW calculator with a PW basis set. Inspiration taken from this example:
https://wiki.fysik.dtu.dk/gpaw/tutorials/bandstructures/bandstructures.html.
'''
```

```
23
24  def convergeK(atoms, tol=1e-4, kstart=4):
25      # Converge total energy by increasing k-space sampling until total energy changes by
26      # <10^-4 eV.
27
28      # DBs
29      convDB = connect('./bulk.db', append=False)  # DB for electronic spectrum
30
31      k = kstart
32      Etot_old = 1
33      Etot_new = 2
34      E = []
35      ks = []
36      i = 1
37      while np.abs(Etot_new - Etot_old) > tol:
38          start = time.time()
39          Etot_old = Etot_new
40          # if world.rank == 0:
41          print(f'---- Iteration: {i} ---- k={k} ----')
42
43          calc = GPAW(
44                  mode=PW(200),                 # cutoff - lower for computational efficiency
45                  kpts=(k, k, k),               # k-points
46                  txt=f'./gpaw-out/k={k}.txt'   # output file
47              )
48          atoms.set_calculator(calc)
49          Etot_new = atoms.get_potential_energy()  # Calculates the total DFT energy of the bulk material
50          end = time.time()
51
52          # if world.rank == 0:
53          print(f'Energy: {Etot_new:.4f} eV ---- Time: {(end-start):.2f} s')
54          E.append(Etot_new)
55          ks.append(k)
56          k += 4
57          i += 1
58      # Save calculator state and write to DB
59      # if world.rank == 0:
60      convDB.write(atoms, data={'energies': E, 'ks': ks})
61      calc.write('kConverge.gpw')
62      print('Written to DB')
63
64      return k, calc
65
66  # Define the Si bulk-structure
67  atoms = bulk('Si', 'diamond', 5.43)
68  if world.rank == 0:
69      print('System created')
70
71  # Find optimal k parameter
72  k, calc = convergeK(atoms, tol=1e-4, kstart=4)
73  if world.rank == 0:
74      print(f'Optimal k-parameter: k={k}')
75
76  # Perform a ground state energy calculation to get the ground state density
77  atoms.get_potential_energy()
78
79  # Save the calculator
80  calc.write('Si_calc.gpw')
81  if world.rank == 0:
```

```
    print('Calculator saved')

#### Electronic band structure
# if world.rank == 0:
# print('Electronic structure calculation started')
# calc = GPAW(
#     'Si_calc.gpw',
#     nbands=16,                              # Include more bands than convergence since metallic
#     fixdensity=True,                        # Fixate the density
#     symmetry='off',                         # Check all points along the path
#     kpts={'path': 'GXWKL', 'npoints': 60},
#     convergence={'bands': 8},
#     txt='Si_calc.txt'
# )
# calc.get_potential_energy()  # Converge the system
# # if world.rank == 0:
# print('Electronic structure converged')

atoms, calc = restart('Si_calc.gpw')
# kpts = {'size': (20,20,20)}
kpts = {'path': 'GXWKL', 'npoints': 60}
calc.set(
    kpts = kpts,
    fixdensity=True,
    symmetry='off',
)

# Fix the potential
calc.get_potential_energy()

# Get band structure and dos
Ebs = atoms.calc.band_structure()  # Get the band structure
if world.rank == 0:
    print('Electronic band structure calculated')

# Set new k-mesh to the calculator to get a nice DOS
kpts = {'size': (28,28,28)}
calc.set(
    kpts = kpts,
    fixdensity=True,
    symmetry='off',
)
# Fix the potential
calc.get_potential_energy()

e, dos = calc.get_dos(spin=0, npts=1001, width=0.2)  # Get energy and density of states
e_f = calc.get_fermi_level()
Edos = {
    'e': e,
    'dos': dos,
    'fermi': e_f
}

# Save results
pickle.dump( Ebs, open( "Ebs.p", "wb" ) )  # Save the electronic band structure
pickle.dump( Edos, open( "Edos.p", "wb" ) )  # Save the electronic DOS
calc.write('Si_electrons.gpw')
if world.rank == 0:
    print('Electronic structure calculation completed')

```

```

#### Phononic band structure
# if world.rank == 0:
print('Phononic structure calculation started')
atoms, calc = restart('Si_calc.gpw')
# kpts = {'size': (20,20,20)}
calc.set(
    symmetry='off',
)


# Set up the ASE phonon calculator
N = 2  # Use a 2x2x2 supercell
ph = Phonons(atoms, calc, supercell=(N, N, N), delta=0.05, name='./phonons/ph_Si')

# Run the phonon calculation
if world.rank == 0:
    print('********* Phonon calculation started *********')
ph.run()
if world.rank == 0:
    print('********* Phonon calculation completed *********')
ph.read(acoustic=True)

# Define BZ-path - use the same as for the electronic calculation
path = atoms.cell.bandpath('GXWKL', npoints=60)

# Fetch band structure and dos
if world.rank == 0:
    print('********* Calculating phononic band structure *********')
Pbs = ph.get_band_structure(path)
if world.rank == 0:
    print('********* Phononic band structure calculated *********')
Pdos = ph.get_dos(kpts=(20, 20, 20)).sample_grid(npts=60, width=1e-3)
if world.rank == 0:
    print('********* Phononic DOS calculated *********')

# Save results
pickle.dump( Pbs, open( "Pbs.p", "wb" ) )  # Save the phononic band structure
pickle.dump( Pdos, open( "Pdos.p", "wb" ) )  # Save the phononic DOS
# calc.write('Si_phonons.gpw') # Don't need to save this calc
if world.rank == 0:
    print('Phononic structure calculation completed')
```

```
# ASE
from ase import Atoms
from ase.dft.bandgap import bandgap
from ase.parallel import world

# GPAW
from gpaw import GPAW, restart


# Restart electronicSi calculator and calculate bandgap

_, calc = restart('Si_electrons.gpw')

# Indirect bandgap
gap, p1, p2 = bandgap(calc, direct=False, output='indirectBandgap.txt')
print(f'Indirect bandgap: {gap:.2f} eV')
gap, p1, p2 = bandgap(calc, direct=True, output='directBandgap.txt')
```

```
print(f'Direct bandgap: {gap:.2f} eV')
```

```
# Internal imports
import pickle

# External imports
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

# ASE
from ase import Atoms
from ase.db import connect


# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

def soundV(bs, modes):
    # Get K-points and energies
    k = bs.path.kpts
    e = bs.energies
    N = 30
    l = 2
    v = 0
    print(k.shape)
    print(e.shape)
    hbar = 6.582e-16 # eVs
    fig, ax = plt.subplots(figsize=(8,6))
    for i in range(modes):
        omegar = e[0,:N,i] / hbar
        kr = k[:N, i%3]
        if i==1 or i==2:
            # The only acoustice branches which goes to 0 are these
            # The last mode is overlayed on one of the others
            # if i==1:
            i_off = len([i for i,k in enumerate(kr) if k <= 0.0 and i>0])   # offset to skip zero indexes
            print(kr[i_off:])
            v = np.abs( (omegar[i_off+l] - omegar[i_off+0]) / (kr[i_off+l] - kr[i_off+0]) ) / 1e10 #   /s
            print(v)
            ax.plot(kr, omegar*hbar, label=f'{i}')
            ax.plot(kr[i_off:i_off+l], omegar[i_off:i_off+l]*hbar, color='r', label=f'{i}')
    ax.legend(loc='best')
    plt.show()

    return None

# Connect to DB
bulkDB = connect('./bulk.db')

# Extract and plot convergence data
fig, ax = plt.subplots(figsize=(8,6))
```

```
dbList = list(bulkDB.select())

ks = dbList[0].data['ks']
E = dbList[0].data['energies']
ax.plot(ks, E)
ax.set_xlabel(r'Number of $k$-points')
ax.set_ylabel('Energy (eV)')
ax.grid()
plt.tight_layout()
plt.savefig('convergenceSi')

################# Extract and plot band electronic band structure and DOS
bs = pickle.load(open( "Ebs.p", "rb" ))
d = pickle.load(open( "Edos.p", "rb" ))

fig, ax = plt.subplots(1,2, figsize=(12,6))

# BS
bs.energies = bs.energies - d['fermi']
emax = 10
emin = -14
bs.plot(filename='', ax=ax[0], show=False, emax=emax, emin=emin)
ax[0].set_ylabel(r'Energy relative to $\epsilon_F$ (eV)')
# lims = (bs.energies.min(), bs.energies.max())
# ax[0].set_ylim(lims)

# DOS
# ax[1].plot(d['e']-d['fermi'], d['dos'])
# ax[1].fill_between( d['dos'], d['e']-d['fermi'], y2=0, color='grey',
#                     edgecolor='k', lw=1)
ax[1].plot(d['dos'], d['e']-d['fermi'])
ax[1].set_xticks([])
# ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
ax[1].set_xlabel(r'DOS')
ax[1].set_ylabel(r'Energy relative to $\epsilon_F$ (eV)')
ax[1].set_xticks([])
ax[1].set_ylim((emin,emax))
plt.tight_layout()
plt.savefig('electronicSi')

################# Extract and plot phonon band structure and DOS
bs = pickle.load(open( "Pbs.p", "rb" ))
dos = pickle.load(open( "Pdos.p", "rb" ))

modes = bs.energies.shape[2]
print(f'Number of phonon modes: {modes}')
# Compute sound velocity
v = soundV(bs, modes)

fig, ax = plt.subplots(1,2, figsize=(12,6))
emax = 0.08
bs.plot(ax=ax[0], emin=0, emax=emax)
ax[0].set_ylabel(r'Energy (eV)')

ax[1].fill_between(dos.weights[0], dos.energy, y2=0, color='grey',
                   edgecolor='k', lw=1)

ax[1].set_ylim(0, emax)
ax[1].set_xticks([])
ax[1].set_ylabel(r'Energy (eV)')
# ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
ax[1].set_xlabel(r'DOS')
```

```
118 plt.savefig('phonon_task7.png')
119 plt.tight_layout()
120 plt.savefig('phononSi')
121 plt.show()
```