

Assignment 5 – Computational Materials and Molecular Physics

Second hand-in

Eric Lindgren – CID: ericlin

March 17, 2020

1 Task 1: Relax nanoparticles

The given nanoparticles were relaxed using the BFGS module in ASE, after which their respective cohesive energies per atom were calculated. These 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)	2.5595	2.7393	2.8473	2.8992	2.9496

Cluster	Al ₉₀	Al ₁₀₈	Al ₁₃₈	Al ₂₀₁	Al ₃₁₆
E_{Coh} (eV)	2.9841	2.0104	3.0416	3.0883	3.1297

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 perimeter 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.

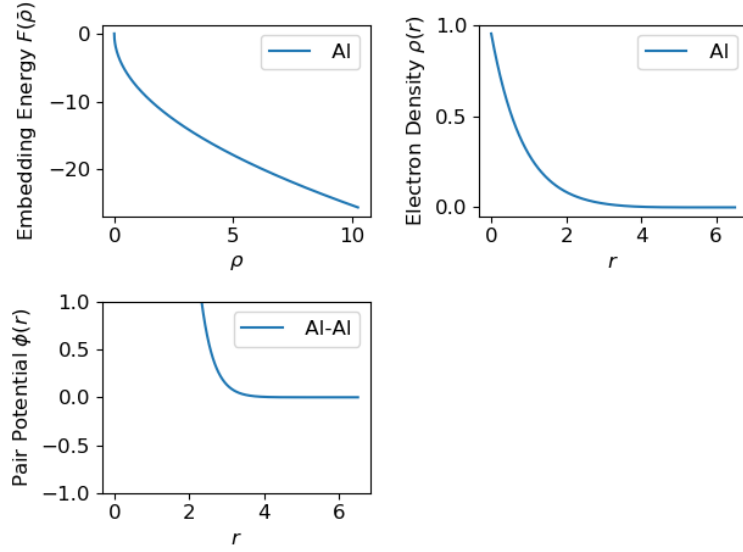


Figure 1: The EAM potential used in task 1. Note that the pair potential seem to have an effective distance of $\sim 4 \text{ \AA}$, which is around the bond length of Al.

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 $\sim 4 \text{ \AA}$, 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 \AA , with a cohesive energy of $E_{Coh} \sim 3.397 \text{ eV}$. This cohesive energy is inline with experimental results, where the lattice parameter is $a = 4.0495 \text{ \AA}$ and the cohesive energy is $E_{Coh} = 3.39 \text{ eV}$ 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.

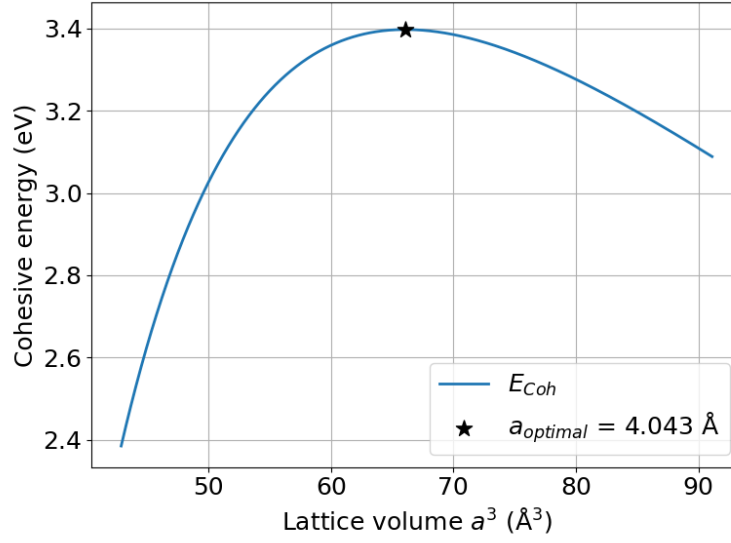


Figure 2: Bulk Al cohesive energy as a function of lattice volume. Note that the maximum of the cohesive energy corresponds to a lattice parameter value of $a \approx 4.043$ Å.

3 Task 3: Vibrational properties of the nanoparticles

The vibrational behaviour for the different nanoparticles were computed by first relaxing the structures and then 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 ($< 10^{-3} \text{ cm}^{-1}$) for all systems except for Al_{13} .

Table 2: Obtained number of modes, imaginary frequencies and zero-frequencies 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.

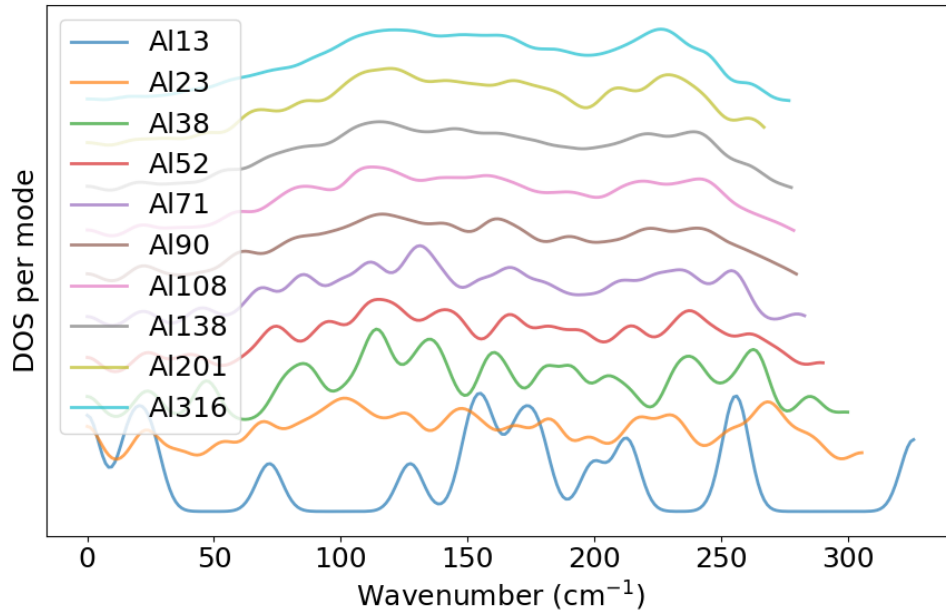


Figure 3: Vibrational DOS for the different nanoclusters. Note that the DOS seems to converge towards a bimodal DOS for larger clusters.

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 $N \rightarrow \infty$ 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 Brillouin 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 ~ 12.0 km/s and ~ 17.3 km/s for the transversal mode and the longitudinal mode, respectively. This is much higher than their experimental values, which should be 3.0 km/s and 6.4 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.

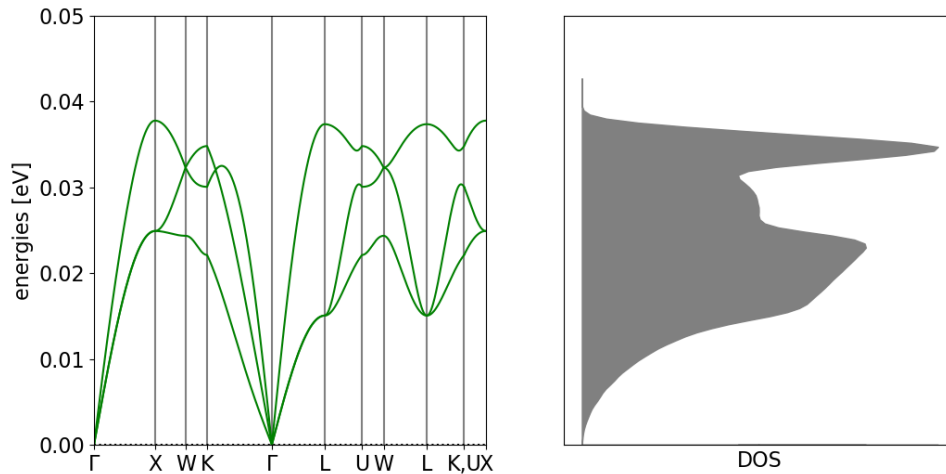


Figure 4: Phonon bandstructure and DOS for bulk Al. Notice that the bandpath has three individual modes, two of which are zero at the Brillouin zone center and that the DOS seems to be what the individual DOS for the nanoparticles converged towards in task 3.

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.

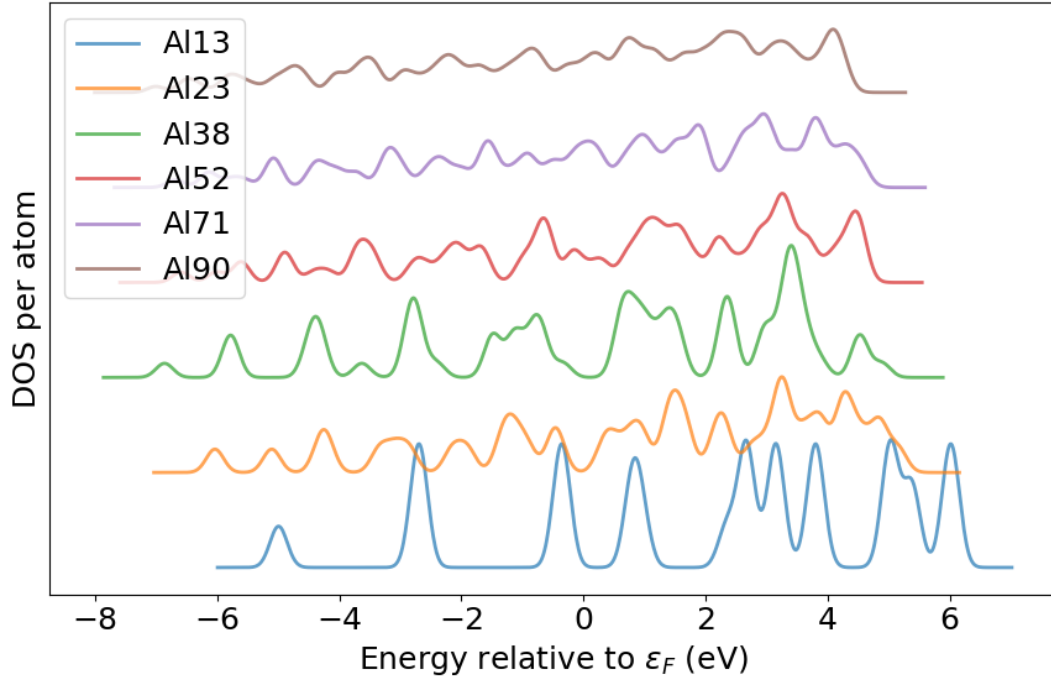


Figure 5: Electronic DOS for the nanoparticles. Note that, as in task 3, the density of states seem to converge to a common shape with increasing cluster size.

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.

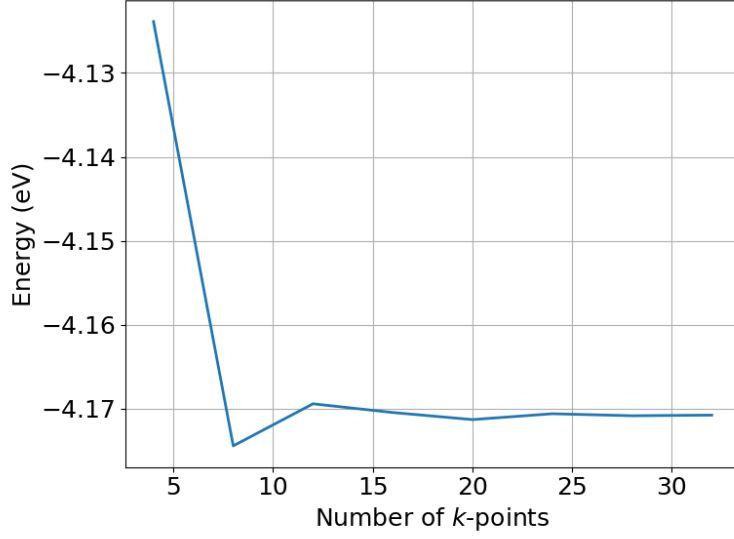


Figure 6: Cluster energy as a function of the number of sampled k -points in each direction.

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. 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 0.1. 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 isn't really too similar to the converged spectrum in task 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 optimisation 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 re-scaling 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.

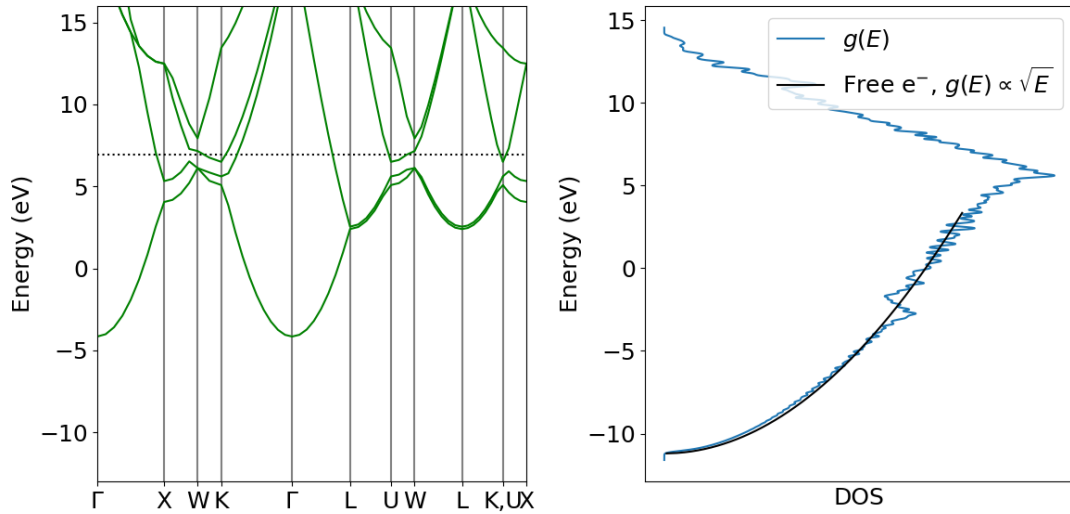


Figure 7: Electronic band structure and DOS for bulk Al. Note that the DOS agrees fairly well with the free electron model's prediction.

7 Task 7: Electrons and phonons in Si

The electronic and vibrational properties of Si were 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].

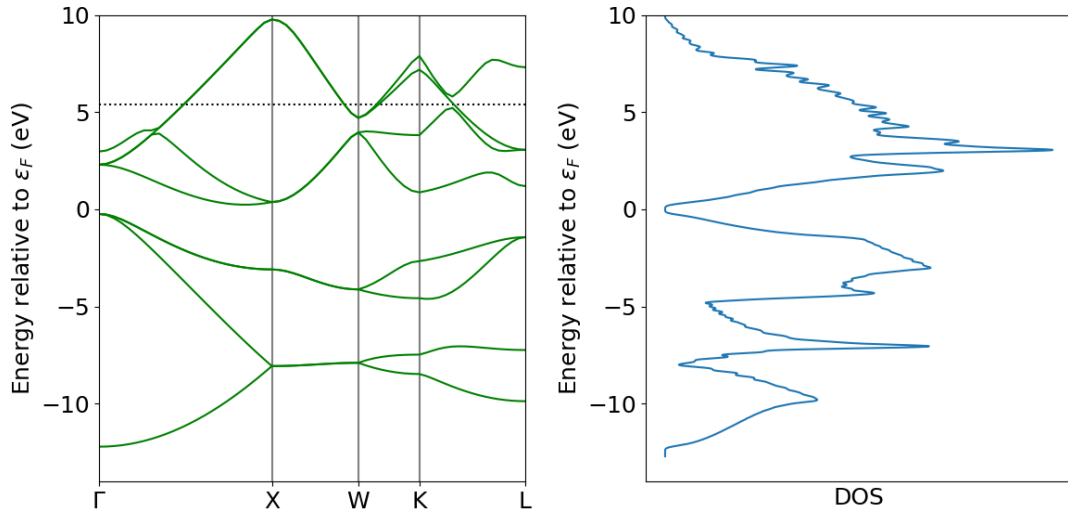


Figure 8: Electronic structure for Si. Disregard the horizontal line in the band spectrum plot; it is an artefact from plotting. Notice the bandgap which is clearly visible in the DOS.

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 ~ 2500 km/s and for the longitudinal branch ~ 19 km/s. This is, as in task 4, very different from the experimental values of 5.5 km/s and 8.4 km/s respectively [7], and like in task 4 I unfortunately don't have the time to investigate this further.

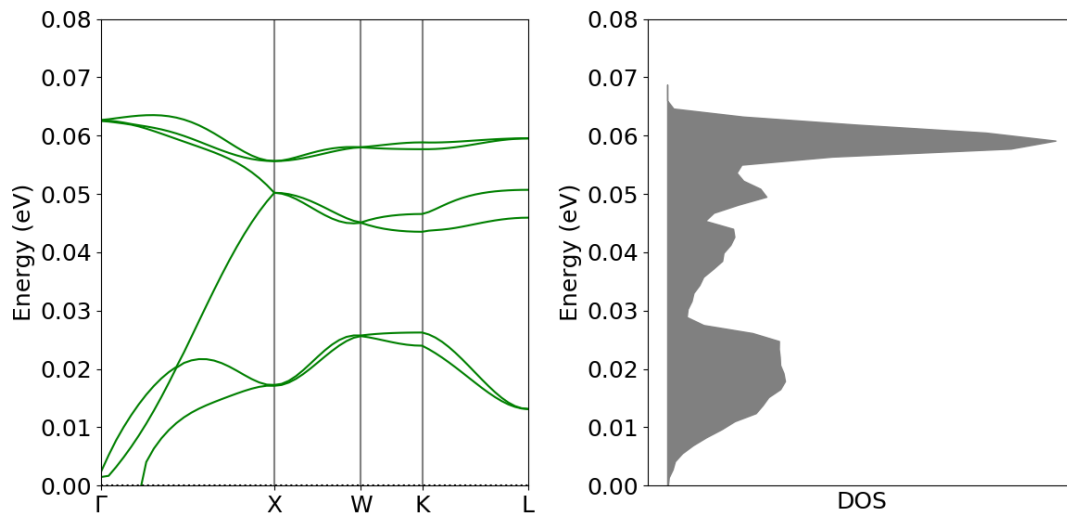


Figure 9: Vibrational structure for Si. Notice the division into three optical and three acoustic phonon modes.

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-AZqFiCLZ4h2Ct1E1KvRvHHDsEzFPM00tJbyqTBvR8hzJ3FRYUGLI540yugUxd9hhGSrMhSlnAzywPhEMACS94UwQ5pZu5z7uLAhQIaNmP9S2_mRhotUGqw2qqYqX5S9jkgX3xHIOfsHTL5wQIEUnAQU_5yqFHMpA_U3zLGBIVQ2U4X03DXEP-WoKSBzkkd1oV0m10SYcoc9-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".

```

1  # External imports
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # ASE
6  from ase import Atoms
7  from ase.db import connect
8  from ase.calculators.eam import EAM
9  from ase.optimize import BFGS
10
11
12 # Set plot params
13 plt.rc('font', size=12)           # controls default text sizes
14 plt.rc('axes', titlesize=12)      # fontsize of the axes title
15 plt.rc('axes', labelsiz=12)       # fontsize of the x and y labels
16 plt.rc('xtick', labelsiz=12)      # fontsize of the tick labels
17 plt.rc('ytick', labelsiz=12)      # fontsize of the tick labels
18 plt.rc('legend', fontsize=12)     # legend fontsize
19
20 def out(s, f):
21     ''' Prints s to console and file f. '''
22     print(s)
23     print(s, file=f)
24
25 # Load the database, row by row, and calculate the cohesive energy.
26 # The cohesive energy is the same as the potential energy since that is
27 # the energy required to separate the atoms.
28 db = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
29 # Sort the clusters based on number of atoms
30 allClust = list(db.select())
31 sort = np.argsort([len(clust.numbers) for clust in allClust])
32 allClust = np.array(allClust)[sort]
33
34 Efile = open('ECoh.txt', 'w')
35 out(s='----- Cluster energies -----', f=Efile)

```

```

36 for clust in allClust:
37     # General info
38     atoms = clust.toatoms()
39     N = len(atoms.positions)
40     # Define calculator
41     mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
42     atoms.set_calculator(mishin)
43     # Relax structure
44     dyn = BFGS(atoms, trajectory=f'traj/Al{N}.traj')
45     dyn.run(fmax=0.05)
46     # Calculate cohesive energy
47     ECoh = np.abs(atoms.get_potential_energy() / N) # Cohesive energy is the ←
         positive potential energy
48
49     # Print results
50     str1 = f'| N={N}'
51     str2 = 'Cohesive energy/atom:'
52     str3 = f' {ECoh:.4f} eV   |'
53     out(str1 + str2.rjust(30-len(str1)) + str3.rjust(35-len(str2)), f=Efile)
54 out('-----', f=Efile)
55
56 # Plot and save a picture of the potential
57 mishin.plot()
58 plt.tight_layout()
59 plt.savefig('al_potential.png')
60 # plt.show()
61
62 # The potential may not be perfectly suited for the smaller nanoparticles, since they←
        don't exhibit the same symmetries as the bulk Al.
63 # Bulk Al is crystalline fcc.

```

A.2 Task 2

Please run in the folder "task2".

```

1  # External imports
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from tqdm import tqdm
5
6  # ASE
7  from ase import Atoms
8  from ase.db import connect
9  from ase.calculators.eam import EAM
10 from ase.build import bulk
11
12
13 # Set plot params
14 plt.rc('font', size=18) # controls default text sizes
15 plt.rc('axes', titlesize=18) # fontsize of the axes title
16 plt.rc('axes', labelsz=18) # fontsize of the x and y labels
17 plt.rc('xtick', labelsz=18) # fontsize of the tick labels
18 plt.rc('ytick', labelsz=18) # fontsize of the tick labels
19 plt.rc('legend', fontsize=18) # legend fontsize
20
21
22 # The true lattice parameter is 4.04 - search around this minimum
23 latParams = np.linspace(3.5, 4.5, 1000)

```

```

24
25 E = []
26 for a in tqdm(latParams):
27     atoms = bulk('Al', 'fcc', a)
28
29     # Calculate potential energy
30     mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
31     atoms.set_calculator(mishin)
32     E.append(atoms.get_potential_energy())
33
34 E = np.abs(E) # Cohesive energy is abs of potential energy
35 # Find maximum cohesive energy and best lattice parameter
36 maxIdx = np.argmax(E)
37 aMax = latParams[maxIdx]
38 EMax = E[maxIdx]
39 print(f'Optimal lattice parameter: {aMax:.3f} , Cohesive energy: {EMax:.3f} eV')
40
41 # Plot results
42 fig, ax = plt.subplots(figsize=(8,6))
43 ax.plot(latParams**3, E, linewidth=2, linestyle='-', marker='o', markersize=0, color='C0', label=r'$E_{Coh}$')
44 ax.scatter(aMax**3, EMax, marker='*', s=150, c='k', zorder=3, label=r'$a_{optimal}$ ' + f'={aMax:.3f} ')
45 ax.set_xlabel(r'Lattice volume $a^3$ ($\rm \AA^3$)')
46 ax.set_ylabel(r'Cohesive energy (eV)')
47 ax.legend(loc='best')
48 ax.grid()
49 plt.tight_layout()
50 plt.savefig('E_task2.png')
51 # plt.show()
52
53 # The lattice parameter agrees with experimental values up to 2 decimal
54 # places, and the cohesive energy seems to match those (per atom)
55 # from task1 somewhat well.
56 # However, the cohesive energy for bulk aluminium averaged over all atoms
57 # is lower, which could be due to the increased stability of the bulk crystal.
58 # In the nanoparticles there are atoms on the surface which are bound less
59 # tight which reduces the average cohesive energy. The effects of these
60 # 'surface atoms' decreases with the size of the nanoparticle which can be
61 # seen by the cohesive energy increasing with the number of atoms in the
62 # nanoparticle. The bulk Al corresponds to the cohesive energy of an infinite
63 # size nanoparticle, i.e. no edge effects.
64
65 # We expect EAM to give good results, since it is particularly good for FCC metals
66 # [https://wiki.fysik.dtu.dk/ase/ase/calculators/eam.html].

```

A.3 Task 3

Please run in the folder "task3".

```

1 # Internal imports
2 import os.path as p
3
4 # External imports
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8

```

```

9  # ASE
10 from ase import Atoms
11 from ase.db import connect
12 from ase.calculators.eam import EAM
13 from ase.vibrations import Vibrations
14 from ase.phonons import Phonons
15 from ase.optimize import BFGS
16
17
18 # Set plot params
19 plt.rc('font', size=18)           # controls default text sizes
20 plt.rc('axes', titlesize=18)      # fontsize of the axes title
21 plt.rc('axes', labelsiz=18)       # fontsize of the x and y labels
22 plt.rc('xtick', labelsiz=18)      # fontsize of the tick labels
23 plt.rc('ytick', labelsiz=18)      # fontsize of the tick labels
24 plt.rc('legend', fontsize=18)     # legend fontsize
25
26 def out(s, f):
27     ''' Prints s to console and file f. '''
28     print(s)
29     print(s, file=f)
30
31 # Load the database, row by row, and calculate the cohesive energy.
32 # The cohesive energy is the same as the potential energy since that is
33 # the energy required to separate the atoms.
34 db = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
35 vibDB = connect('./vib.db', append=False)
36 # Sort the clusters based on number of atoms
37 allClust = list(db.select())
38 sort = np.argsort([len(clust.numbers) for clust in allClust])
39 allClust = np.array(allClust)[sort]
40
41 for clust in allClust:
42     # General info
43     atoms = clust.toatoms()
44     N = len(atoms.positions)
45     # Define calculator - mishin
46     mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
47     atoms.set_calculator(mishin)
48     # Relax structure
49     dyn = BFGS(atoms, trajectory=f'traj/Al{N}.traj')
50     dyn.run(fmax=0.05)
51     # Get vibrational spectrum
52     str1 = f'----- N={N}'
53     str2 = '          '
54     print(str1 + str2.ljust(40-len(str1)))
55
56     ##### Using Vibrations module
57     v = Vibrations(atoms, name=f'./vibs/vib_{N}')
58     if not p.isfile(f'./vibs/vib_{N}.all.pckl'):
59         print('Running vibration calculation')
60         v.run()
61         v.combine() # Combine pickle files
62         # Get frequencies and DOS - i.e # of states per frequency
63         all_freq = v.get_frequencies()
64         # if N==38:
65         #     print(v.summary())
66         #     print(all_freq)
67         (freq, counts) = np.unique(all_freq, return_counts=True)
68         fold_freq = v.fold(np.real(freq), np.real(counts), start=0, end=np.real(freq.max←
69             ()), width=12, normalize=False)
70         f_freq = np.array(fold_freq[0])

```

```

70     f_dos = np.array(fold_freq[1])
71     freq = np.array(freq)
72     dos = np.array(counts)
73
74     # Get number of modes
75     # e = False
76     # i=0
77     # modes = 0
78     # while not e:
79     #     try:
80     #         v.get_mode(i)
81     #         modes += 1
82     #         i += 1
83     #     except:
84     #         e = True
85
86     # Save to db
87     vibDB.write(atoms, data={'frequency': freq, 'DOS': dos, 'f_freq': f_freq, 'f_dos': ←
88         : f_dos})
89
90     ##### Using Phonons module
91     # ph = Phonons(atoms, mishin, delta=0.05, name='./phonons/ph_Si')
92     # ph.run()
93     # # ph.combine()
94     # ph.read(acoustic=True)
95     # Pdos = ph.get_dos(kpts=(20, 20, 20)).sample_grid(npts=60, width=1e-3)
96     # print(Pdos)
97     print('Sucessfully saved all data to DB')

```

```

1  # External imports
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  from tqdm import tqdm
6
7  # ASE
8  from ase import Atoms
9  from ase.db import connect
10
11
12  # Set plot params
13  plt.rc('font', size=18)           # controls default text sizes
14  plt.rc('axes', titlesize=18)      # fontsize of the axes title
15  plt.rc('axes', labelsiz=18)       # fontsize of the x and y labels
16  plt.rc('xtick', labelsiz=18)      # fontsize of the tick labels
17  plt.rc('ytick', labelsiz=18)      # fontsize of the tick labels
18  plt.rc('legend', fontsize=18)     # legend fontsize
19
20  def out(s, f):
21      ''' Prints s to console and file f. '''
22      print(s)
23      print(s, file=f)
24
25  # Read DB
26  vibDB = connect('./vib.db')
27
28  # Plot vibrational spectra
29  vFile = open('vib.txt', 'w')
30  out(s='----- Cluster Vibrations ←

```

```

31 -----', f=vFile)
32 fig, ax = plt.subplots(figsize=(9,6))
33 for i, clust in enumerate(vibDB.select()):
34     atoms = clust.toatoms()
35     freq = clust.data['frequency']
36     dos = clust.data['DOS']
37     # Extract info
38     N = len(atoms.positions)
39     nModes = len(freq)
40     isC = np.iscomplex(freq).sum()
41     isZ = len(freq) - np.count_nonzero(np.real(freq))
42     # Print info
43     str1 = f'| Al{N} '
44     str2 = 'Modes: '
45     str3 = f'{nModes} '
46     str4 = '    Imaginary frequencies: '
47     str5 = f'{isC} '
48     str6 = '    Zero frequencies: '
49     str7 = f'{isZ} |'
50     out(str1 + str2.rjust(20-len(str1)) + str3.rjust(15-len(str2)) + str4 + str5 + ↵
51         str6 + str7, f=vFile)
52
53 # Plot
54 f_freq = clust.data['f_freq']
55 f_dos = clust.data['f_dos']
56 f_freq = np.real(f_freq)
57 # ax.axhline(0.05*i, color='k', alpha=0.2)
58 ax.plot(f_freq, f_dos/nModes + 0.05*i, alpha=0.7, linewidth=2, label=f'Al{N}')
59 # ax.grid()
60 ax.legend(loc='upper left')
61 ax.set_xlabel(r'Wavenumber ($\rm cm^{-1}$)')
62 # ax.set_ylabel(r'DOS per mode ($\rm cm$)')
63 ax.set_ylabel(r'DOS per mode')
64 ax.set_yticks([])
65 # ax.set_zlabel(r'DOS ($\rm cm$)', labelpad=10) # TODO set proper units
66 # ax.set_zlabel(r'DOS')
67 plt.tight_layout()
68 plt.savefig('figTask3.png')
69 out(s='↵
    -----', f=vFile)

```

A.4 Task 4

Please run in the folder "task4".

```

1 # Internal imports
2 import os.path as p
3
4 # External imports
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8
9 # ASE
10 from ase import Atoms
11 from ase.db import connect
12 from ase.calculators.eam import EAM
13 from ase.build import bulk

```



```

14 from ase.vibrations import Vibrations
15 from ase.phonons import Phonons
16 from ase.visualize import view
17
18
19 # Set plot params
20 plt.rc('font', size=16)           # controls default text sizes
21 plt.rc('axes', titlesize=16)      # fontsize of the axes title
22 plt.rc('axes', labelsiz=16)       # fontsize of the x and y labels
23 plt.rc('xtick', labelsiz=16)      # fontsize of the tick labels
24 plt.rc('ytick', labelsiz=16)      # fontsize of the tick labels
25 plt.rc('legend', fontsize=16)     # legend fontsize
26
27 def soundV(bs, modes):
28     # Get K-points and energies
29     k = bs.path.kpts
30     e = bs.energies
31     N = 20
32     v = 0
33     hbar = 6.582e-16 # eVs
34     fig, ax = plt.subplots(figsize=(8,6))
35     for i in range(modes):
36         omegar = e[0,:N,i] / hbar
37         kr = k[:N,i]
38         if not kr[-1] == 0:
39             # The last mode is overlayed on one of the others
40             v = np.abs( (omegar[-1] - omegar[0]) / (kr[-1] - kr[0]) ) / 1e10 # /s
41             print(v)
42             ax.plot(k[:N,i], e[0,:N,i])
43     return None
44
45 # DBs
46 vibDB = connect('./vib.db', append=False) # DB for vibration spectrum
47 phonDB = connect('./ph.db', append=False) # DB for phonon structure and DOS
48 # Using optimal lattice parameter from task 2
49 a = 4.043 # A
50 atoms = bulk('Al', 'fcc', a)
51 # view(atoms) # DEBUG
52
53 ##### Calculate vibrational spectrum
54 # Attach EAM calculator to atoms
55 mishin = EAM(potential='../CourseGitRepo/HA5_al_potential.alloy') # Set up EAM
56 atoms.set_calculator(mishin)
57
58 # Get vibrational spectrum
59 v = Vibrations(atoms, name='./vibs/vib_bulk')
60 v.run()
61 v.summary()
62 # Get frequencies and DOS - i.e # of states per frequency
63 (freq, counts) = np.unique(v.get_frequencies(), return_counts=True)
64 freq = np.array(freq)
65 dos = np.array(counts)
66 # Save to db
67 vibDB.write(atoms, data={'frequency': freq, 'DOS': dos})
68
69 ##### Calculate band structure
70 N=7 # Use a supercell of size 7x7x7
71 ph = Phonons(atoms, mishin, name='./phonons/ph_bulk', supercell=(N,N,N), delta=0.05)
72 ph.run()
73 # Read the results from the run and obtain the bandpath and DOS
74 ph.read(acoustic=True)
75 # ph.clean()

```

```

76
77 # lat.plot_bz(show=True) # Visualize Brillouin zone
78 # Al has Space Group 225 [https://materialsproject.org/materials/mp-134/]
79 # from which Bilbao Cryst gives us
80 # Also here is given optimal vectors https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.↵
    html
81 # Use default path for now
82 # And here [https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.html#ase.dft.band_structure↵
    .BandStructure]
83 path = atoms.cell.bandpath(path='GXWGLUWLK,UX', density=100)
84
85 bs = ph.get_band_structure(path)
86 modes = bs.energies.shape[2]
87 print(f'Number of phonon modes: {modes}')
88
89 # Compute sound velocity
90 v = soundV(bs, modes)
91
92 dos = ph.get_dos(kpts=(20,20,20)).sample_grid(npts=100, width=1e-3)
93
94 # Plot phonon spectrum and DOS
95 fig, ax = plt.subplots(1,2, figsize=(12,6))
96 emax = 0.05
97 bs.plot(ax=ax[0], emin=0.0, emax=emax)
98
99 # ax[1] = fig.add_axes([.8, .07, .17, .85])
100 ax[1].fill_between(dos.weights[0], dos.energy, y2=0, color='grey',
101                   edgecolor='k', lw=1)
102
103 ax[1].set_ylim(0, emax)
104 ax[1].set_yticks([])
105 ax[1].set_xticks([])
106 # ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
107 ax[1].set_xlabel(r'DOS') # TODO set proper units
108
109 plt.savefig('phonon_task4.png')
110 plt.tight_layout()

```

A.5 Task 5

Please run in the folder "task5".

```

1 # Internal imports
2 import time
3 import pickle
4
5 # External imports
6 import numpy as np
7
8 # ASE
9 from ase import Atoms
10 from ase.db import connect
11 from ase.dft.dos import DOS
12 from ase.parallel import world
13 from ase.optimize import BFGS
14
15 # GPAW
16 from gpaw import GPAW, PW

```

```

17
18 '''
19 Perform GPAW DFT calculation of the electron density of states
20 for the nanoparticles with N < 100.
21 '''
22 # Connect to DB
23 structDB = connect('../CourseGitRepo/HA5_Al-clusters-initial.db')
24 eosDB = connect('./eos.db', append=False)
25
26 # Sort the clusters based on number of atoms
27 allClust = list(structDB.select())
28 sort = np.argsort([len(clust.numbers) for clust in allClust])
29 allClust = np.array(allClust)[sort]
30
31 for clust in allClust:
32     # General info
33     atoms = clust.toatoms()
34     N = len(atoms.positions)
35     if(N<100):
36         start = time.time()
37         # if world.rank == 0:
38         print(f'Calculating EOS for Al{N}')
39
40         # Define electron calculator (GPAW)
41         calc = GPAW(
42             mode=PW(300), # Lower for computational efficiency
43             txt=f'./gpaw-out/EOS_{N}_1core.txt'
44         ) # Use the same calculator as in task6
45         atoms.set_calculator(calc)
46         pot_e = atoms.get_potential_energy() # Self-consistently optimize the ↔
47         # electron density
48         # if world.rank == 0:
49         print(f'Cluster Al{N} finished potential energy per atom: {pot_e / N:.2f} eV'↔
50             )
51
52         # Get the electronic DOS
53         dos = DOS(calc, npts=800, width=0.2)
54
55         e = dos.get_energies()
56         d = dos.get_dos()
57         e_f = calc.get_fermi_level()
58         e -= e_f # Subtract the Fermi level from the energy
59
60         ##### Get the DOS using the same method as in task6
61         # print('Electronic band structure calculated')
62         # kpts = {'size': (40,40,40)}
63         # calc.set(
64         #     kpts = kpts,
65         #     fixdensity=True,
66         #     symmetry='off',
67         # )
68         # # Fix the potential
69         # calc.get_potential_energy()
70         # e, dos = calc.get_dos(spin=0, npts=1001, width=0.5) # Get energy and ↔
71         # density of states
72         # e_f = calc.get_fermi_level()
73
74         # Edos = {
75         #     'e': e,
76         #     'dos': dos,
77         #     'fermi': e_f
78         # }

```

```

76
77     # # Save results
78     # pickle.dump( Edos, open( f'./dos/Edos_Al{N}_lcore.p', "wb" ) ) # Save the ←
       electronic DOS
79
80     end = time.time()
81     # if world.rank == 0:
82     print(f'Cluster Al{N} finished ---- Time: {(end-start):.2f} s')
83     eosDB.write(atoms, data={'energy': e, 'DOS': d, 'fermi': e_f})
84     calc.write(f'./calculators/calc{N}.gpw') # Save the calculator
85 else:
86     # if world.rank == 0:
87     print(f'Skipping Al{N}')

```

```

1  # External imports
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  from tqdm import tqdm
6
7  # ASE
8  from ase import Atoms
9  from ase.db import connect
10
11
12  # Set plot params
13  plt.rc('font', size=18) # controls default text sizes
14  plt.rc('axes', titlesize=18) # fontsize of the axes title
15  plt.rc('axes', labelsz=18) # fontsize of the x and y labels
16  plt.rc('xtick', labelsz=18) # fontsize of the tick labels
17  plt.rc('ytick', labelsz=18) # fontsize of the tick labels
18  plt.rc('legend', fontsize=18) # legend fontsize
19
20  # Connect to DB
21  eosDB = connect('./eos.db')
22
23  # Extract and plot convergence data
24  # fig, ax = plt.subplots(figsize=(8,6))
25
26  fig, ax = plt.subplots(figsize=(9,6))
27  dbList = list(eosDB.select())
28
29  for i,row in enumerate(dbList):
30      atoms = row.toatoms()
31      N = len(atoms.positions)
32      dos = row.data['DOS']
33      e = row.data['energy'] - row.data['fermi']
34      # ax.axhline(0.5*i, color='k', alpha=0.2)
35      ax.plot(e, dos/N + 0.5*i, alpha=0.7, linewidth=2, label=f'Al{N}')
36      # ax.plot(e, N*np.ones(len(e)), dos, alpha=1-i*0.1, linewidth=2, label=f'Al{N}')
37
38  ax.legend(loc='upper left')
39  ax.set_xlabel(r'Energy relative to  $\epsilon_F$  (eV)')
40  # ax.set_ylabel(r'Cluster size  $N$ ', labelpad=15)
41  # ax.set_zlabel(r'DOS ( $\text{rm eV}^{-1}$ )', labelpad=10) # TODO set proper units
42  ax.set_ylabel(r'DOS per atom')
43  ax.set_yticks([])
44  plt.tight_layout()
45  plt.savefig('dos_task5')
46  plt.show()

```

A.6 Task 6

Please run in the folder "task6".

```
1 # Internal imports
2 import time
3 import pickle
4
5 # External imports
6 import numpy as np
7
8 # ASE
9 from ase import Atoms
10 from ase.db import connect
11 from ase.build import bulk
12 from ase.parallel import world
13 from ase.dft.dos import DOS
14
15 # GPAW
16 from gpaw import GPAW, PW, restart
17
18 '''
19 Perform GPAW DFT calculation for the electronic structure of
20 bulk Al. First find converge number of k-points. Then, converge
21 density self-consistently.
22 '''
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(300), # cutoff
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
50         # the bulk material
51         end = time.time()
```

```

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     return k, calc
64
65
66 # Using optimal lattice parameter from task 2
67 a = 4.043 # A
68 atoms = bulk('Al', 'fcc', a)
69
70 # Find optimal k parameter
71 k, calc = convergeK(atoms, tol=1e-4, kstart=4)
72 print(f'Optimal k-parameter: k={k}')
73
74 # Perform a ground state energy calculation to get the ground state density
75 atoms.get_potential_energy()
76
77 # Save the calculator
78 calc.write('Al_calc.gpw')
79 # if world.rank == 0:
80 print('Calculator saved')
81
82 ##### Electronic band structure
83 # if world.rank == 0:
84 print('Electronic structure calculation started')
85 atoms, calc = restart('Al_calc.gpw')
86 # kpts = {'size': (60,60,60), 'path': 'GXWGLUWLK,UX'}
87 kpts = {'path': 'GXWGLUWLK,UX', 'npoints': 60}
88 calc.set(kpts = kpts, fixdensity=True, symmetry='off')
89
90 # calc = GPAW(
91 #     'Al_calc.gpw',
92 #     nbands=16, # Include more bands than convergence ↔
93 #     since metallic
94 #     fixdensity=True, # Fixate the density
95 #     symmetry='off', # Check all points along the path
96 #     kpts={'path': 'GXWKL', 'npoints': 60},
97 #     convergence={'bands': 8},
98 #     txt='Al_calc.txt'
99 # )
100 # calc.get_potential_energy() # Converge the system
101 # # if world.rank == 0:
102 # print('Electronic structure converged')
103
104 # Fix the potential
105 calc.get_potential_energy()
106
107 # Get band structure and dos
108 Ebs = atoms.calc.band_structure() # Get the band structure
109
110 # if world.rank == 0:
111 print('Electronic band structure calculated')
112 kpts = {'size': (40,40,40)}
113 calc.set(

```

```

113     kpts = kpts,
114     fixdensity=True,
115     symmetry='off',
116 )
117 # Fix the potential
118 calc.get_potential_energy()
119
120 # e, dos = calc.get_dos(spin=0, npts=1001, width=0.5) # Get energy and density of ↔
121 # states
122 dos = DOS(calc, npts=2000, width=0.1)
123 d = dos.get_dos()
124 e = dos.get_energies()
125 f = calc.get_fermi_level()
126 print('Electronic DOS computed')
127 e_f = calc.get_fermi_level()
128 Edos = {
129     'e': e,
130     'dos': d,
131     'fermi': e_f
132 }
133
134 # Save results
135 pickle.dump( Ebs, open( "Ebs.p", "wb" ) ) # Save the electronic band structure
136 pickle.dump( Edos, open( "Edos.p", "wb" ) ) # Save the electronic DOS
137 # if world.rank == 0:
138 print('Electronic structure calculation completed')

```

```

1 # Internal imports
2 import pickle
3
4 # External imports
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8
9 # ASE
10 from ase import Atoms
11 from ase.db import connect
12
13
14 # Set plot params
15 plt.rc('font', size=18) # controls default text sizes
16 plt.rc('axes', titlesize=18) # fontsize of the axes title
17 plt.rc('axes', labelsize=18) # fontsize of the x and y labels
18 plt.rc('xtick', labelsize=18) # fontsize of the tick labels
19 plt.rc('ytick', labelsize=18) # fontsize of the tick labels
20 plt.rc('legend', fontsize=18) # legend fontsize
21
22 # Connect to DB
23 bulkDB = connect('./bulk.db')
24
25 # Extract and plot convergence data
26 fig, ax = plt.subplots(figsize=(8,6))
27 dbList = list(bulkDB.select())
28
29 ks = dbList[0].data['ks']
30 E = dbList[0].data['energies']
31 ax.plot(ks, E, linewidth=2)
32
33 ax.set_xlabel(r'Number of $k$-points')

```

```

34 ax.set_ylabel('Energy (eV)')
35 ax.grid()
36 plt.tight_layout()
37 plt.savefig('kConvergence.png')
38
39 # Extract and plot band electronic band structure and DOS
40 bs = pickle.load(open( "Ebs.p", "rb" ))
41 d = pickle.load(open( "Edos.p", "rb" ))
42
43 fig, ax = plt.subplots(1,2, figsize=(12,6))
44
45 # BS
46 # bs.energies = bs.energies - d['fermi']
47 emax = 16
48 emin = -13
49 bs.plot(filename='', ax=ax[0], show=False, emax=emax, emin=emin)
50 ax[0].set_ylabel(r'Energy (eV)')
51
52 # DOS
53 # e = d['e']-d['fermi']
54 e = d['e']
55 # ax[1].fill_between( d['dos'], e, y2=0, color='grey',
56 #                    edgecolor='k', lw=1, alpha=0.6, label=r'$g(E)$')
57 ax[1].plot(d['dos'], e, label=r'$g(E)$' )
58
59 ePos = np.array([ ei for ei in e if ei>=0 ])
60 # Calculate free electron density
61 # Na = 6.02214076e23 # Avogadro's constant
62 # Z = 3 # Nbr of valence electrons of Al
63 # rho = 2720 # kg/m3
64 # ma = 26.98 * 1.66e-27
65 # n = Na*Z*rho/ma
66 # freeE = 1.5 * n/d['fermi'] * np.sqrt(ePos/d['fermi'])
67 freeE = 0.061*np.sqrt(ePos) # TODO set proper scale
68 ax[1].plot(freeE, ePos-11.2, color='k', label=r'Free $e^-$, $g(E) \propto \sqrt{E}$')
69
70 ax[1].legend(loc='best')
71 ax[1].set_xticks([])
72 # ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$')) # TODO set proper units
73 ax[1].set_xlabel(r'DOS')
74 ax[1].set_ylabel(r'Energy (eV)')
75 # ax[1].set_ylim((emin, emax))
76 plt.tight_layout()
77 plt.savefig('electronicAl.png')

```

A.7 Task 7

Please run in the folder "task7".

```

1 # Internal imports
2 import time
3 import pickle
4
5 # External imports
6 import numpy as np
7
8 # ASE

```



```

9 from ase import Atoms
10 from ase.db import connect
11 from ase.build import bulk
12 from ase.phonons import Phonons
13 from ase.parallel import world
14
15 # GPAW
16 from gpaw import GPAW, PW, restart
17
18 '''
19 Perform DFT calculation for electronic and phononic band structures and density of ←
    states for Si.
20 Uses a GPAW calculator with a PW basis set. Inspiration taken from this example:
21 https://wiki.fysik.dtu.dk/gpaw/tutorials/bandstructures/bandstructures.html.
22 '''
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 print(k)
76
77 # Perform a ground state energy calculation to get the ground state density
78 atoms.get_potential_energy()
79
80 # Save the calculator
81 calc.write('Si_calc.gpw')
82 if world.rank == 0:
83     print('Calculator saved')
84
85 ##### Electronic band structure
86 if world.rank == 0:
87     print('Electronic structure calculation started')
88 calc = GPAW(
89     'Si_calc.gpw',
90     nbands=16, # Include more bands than convergence ←
91     fixdensity=True, # Fixate the density
92     symmetry='off', # Check all points along the path
93     kpts={'path': 'GXWKL', 'npoints': 60},
94     convergence={'bands': 8},
95     txt='Si_calc.txt'
96 )
97 calc.get_potential_energy() # Converge the system
98 # if world.rank == 0:
99 print('Electronic structure converged')
100
101 atoms, calc = restart('Si_calc.gpw')
102 # kpts = {'size': (20,20,20)}
103 kpts = {'path': 'GXWKL', 'npoints': 60}
104 calc.set(
105     kpts = kpts,
106     fixdensity=True,
107     symmetry='off',
108 )
109
110 # Fix the potential
111 calc.get_potential_energy()
112
113 # Get band structure and dos
114 Ebs = atoms.calc.band_structure() # Get the band structure
115 if world.rank == 0:
116     print('Electronic band structure calculated')
117
118 # Set new k-mesh to the calculator to get a nice DOS
119 kpts = {'size': (28,28,28)}
120 calc.set(
121     kpts = kpts,
122     fixdensity=True,
123     symmetry='off',
124 )
125 # Fix the potential
126 calc.get_potential_energy()
127

```

```

128 e, dos = calc.get_dos(spin=0, npts=1001, width=0.2) # Get energy and density of states
129 e_f = calc.get_fermi_level()
130 Edos = {
131     'e': e,
132     'dos': dos,
133     'fermi': e_f
134 }
135
136 # Save results
137 pickle.dump( Ebs, open( "Ebs.p", "wb" ) ) # Save the electronic band structure
138 pickle.dump( Edos, open( "Edos.p", "wb" ) ) # Save the electronic DOS
139 calc.write('Si_electrons.gpw')
140 if world.rank == 0:
141     print('Electronic structure calculation completed')
142
143 ##### Phononic band structure
144 # if world.rank == 0:
145 # print('Phononic structure calculation started')
146 atoms, calc = restart('Si_calc.gpw')
147 # # kpts = {'size': (20,20,20)}
148 calc.set(
149     symmetry='off',
150 )
151
152 # Set up the ASE phonon calculator
153 N = 3 # Use a 2x2x2 supercell
154 ph = Phonons(atoms, calc, supercell=(N, N, N-1), delta=0.05, name='./phonons/ph_Si')
155
156 # Run the phonon calculation
157 if world.rank == 0:
158     print('***** Phonon calculation started *****')
159 ph.run()
160 if world.rank == 0:
161     print('***** Phonon calculation completed *****')
162 ph.read(acoustic=True)
163
164 # Define BZ-path - use the same as for the electronic calculation
165 path = atoms.cell.bandpath('GXWKL', npoints=100)
166
167 # Fetch band structure and dos
168 if world.rank == 0:
169     print('***** Calculating phononic band structure *****')
170 Pbs = ph.get_band_structure(path)
171 if world.rank == 0:
172     print('***** Phononic band structure calculated *****')
173 Pdos = ph.get_dos(kpts=(20, 20, 20)).sample_grid(npts=100, width=1e-3)
174 if world.rank == 0:
175     print('***** Phononic DOS calculated *****')
176
177 # Save results
178 pickle.dump( Pbs, open( "Pbs.p", "wb" ) ) # Save the phononic band structure
179 pickle.dump( Pdos, open( "Pdos.p", "wb" ) ) # Save the phononic DOS
180 # calc.write('Si_phonons.gpw') # Don't need to save this calc
181 if world.rank == 0:
182     print('Phononic structure calculation completed')
183

```

```

1 # ASE
2 from ase import Atoms

```

```

3 from ase.dft.bandgap import bandgap
4 from ase.parallel import world
5
6 # GPAW
7 from gpaw import GPAW, restart
8
9
10 # Restart electronicSi calculator and calculate bandgap
11
12 _, calc = restart('Si_electrons.gpw')
13
14 # Indirect bandgap
15 gap, p1, p2 = bandgap(calc, direct=False, output='indirectBandgap.txt')
16 print(f'Indirect bandgap: {gap:.2f} eV')
17 gap, p1, p2 = bandgap(calc, direct=True, output='directBandgap.txt')
18 print(f'Direct bandgap: {gap:.2f} eV')

```

```

1 # Internal imports
2 import pickle
3
4 # External imports
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8
9 # ASE
10 from ase import Atoms
11 from ase.db import connect
12
13
14 # Set plot params
15 plt.rc('font', size=18) # controls default text sizes
16 plt.rc('axes', titlesize=18) # fontsize of the axes title
17 plt.rc('axes', labelsz=18) # fontsize of the x and y labels
18 plt.rc('xtick', labelsz=18) # fontsize of the tick labels
19 plt.rc('ytick', labelsz=18) # fontsize of the tick labels
20 plt.rc('legend', fontsize=18) # legend fontsize
21
22 def soundV(bs, modes):
23     # Get K-points and energies
24     k = bs.path.kpts
25     e = bs.energies
26     N = 30
27     l = 2
28     v = 0
29     print(k.shape)
30     print(e.shape)
31     hbar = 6.582e-16 # eVs
32     fig, ax = plt.subplots(figsize=(8,6))
33     for i in range(modes):
34         omegar = e[:,N,i] / hbar
35         kr = k[:,N, i%3]
36         if i==1 or i==2:
37             # The only acoustice branches which goes to 0 are these
38             # The last mode is overlayed on one of the others
39             # if i==1:
40             i_off = len([i for i,k in enumerate(kr) if k <= 0.0 and i>0]) # offset ←
41             # to skip zero indexes
42             print(kr[i_off:])
43             v = np.abs( (omegar[i_off+1] - omegar[i_off+0]) / (kr[i_off+1] - kr[i_off+0]) )

```

```

+0]) ) / 1e10 # /s
43     print(v)
44     ax.plot(kr, omegar*hbar, label=f'{i}')
45     ax.plot(kr[i_off:i_off+1], omegar[i_off:i_off+1]*hbar, color='r', label=f'←
        {i}')
46     ax.legend(loc='best')
47     plt.show()
48
49     return None
50
51 # Connect to DB
52 bulkDB = connect('./bulk.db')
53
54 # Extract and plot convergence data
55 fig, ax = plt.subplots(figsize=(8,6))
56 dbList = list(bulkDB.select())
57
58 ks = dbList[0].data['ks']
59 E = dbList[0].data['energies']
60 ax.plot(ks, E)
61 ax.set_xlabel(r'Number of $k$-points')
62 ax.set_ylabel('Energy (eV)')
63 ax.grid()
64 plt.tight_layout()
65 plt.savefig('convergenceSi')
66
67 ##### Extract and plot band electronic band structure and DOS
68 bs = pickle.load(open( "Ebs.p", "rb" ))
69 d = pickle.load(open( "Edos.p", "rb" ))
70
71 fig, ax = plt.subplots(1,2, figsize=(12,6))
72
73 # BS
74 bs.energies = bs.energies - d['fermi']
75 emax = 10
76 emin = -14
77 bs.plot(filename='', ax=ax[0], show=False, emax=emax, emin=emin)
78 ax[0].set_ylabel(r'Energy relative to $\epsilon_F$ (eV)')
79 # lims = (bs.energies.min(), bs.energies.max())
80 # ax[0].set_ylim(lims)
81
82 # DOS
83 # ax[1].plot(d['e']-d['fermi'], d['dos'])
84 # ax[1].fill_between( d['dos'], d['e']-d['fermi'], y2=0, color='grey',
85 #                    edgecolor='k', lw=1)
86 ax[1].plot(d['dos'], d['e']-d['fermi'])
87 ax[1].set_xticks([])
88 # ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
89 ax[1].set_xlabel(r'DOS')
90 ax[1].set_ylabel(r'Energy relative to $\epsilon_F$ (eV)')
91 ax[1].set_xticks([])
92 ax[1].set_ylim((emin,emax))
93 plt.tight_layout()
94 plt.savefig('electronicSi')
95
96 ##### Extract and plot phonon band structure and DOS
97 bs = pickle.load(open( "Pbs.p", "rb" ))
98 dos = pickle.load(open( "Pdos.p", "rb" ))
99
100 modes = bs.energies.shape[2]
101 print(f'Number of phonon modes: {modes}')
102 # Compute sound velocity

```

```

103 # v = soundV(bs, modes)
104
105 fig, ax = plt.subplots(1,2, figsize=(12,6))
106 emax = 0.08
107 bs.plot(ax=ax[0], emin=0, emax=emax)
108 ax[0].set_ylabel(r'Energy (eV)')
109
110 ax[1].fill_between(dos.weights[0], dos.energy, y2=0, color='grey',
111                  edgecolor='k', lw=1)
112
113 ax[1].set_ylim(0, emax)
114 ax[1].set_xticks([])
115 ax[1].set_ylabel(r'Energy (eV)')
116 # ax[1].set_xlabel(r'DOS ($\rm eV^{-1}$)') # TODO set proper units
117 ax[1].set_xlabel(r'DOS')
118 plt.savefig('phonon_task7.png')
119 plt.tight_layout()
120 plt.savefig('phononSi')
121 plt.show()

```