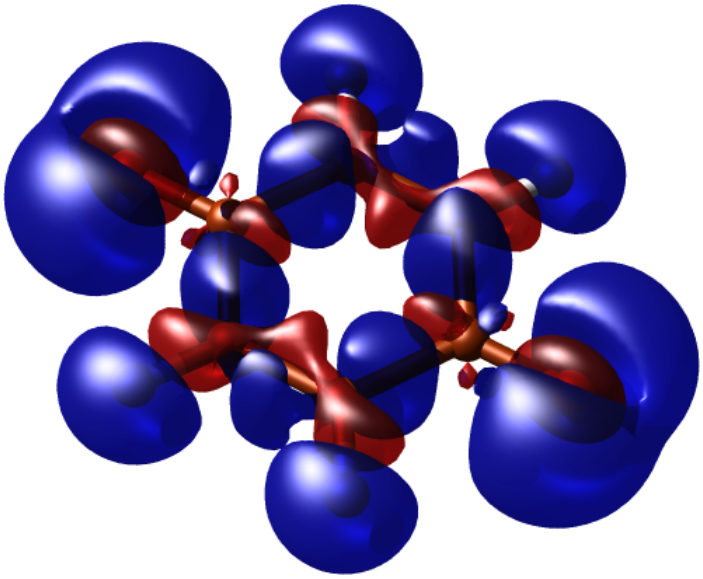
FHI-aims Tutorial

Basics of Electronic-Structure Theory with FHI-aims: Atoms and Molecules



Tutorial I and II: Basics of Electronic-Structure Theory Manuscript for Exercise Problems

Prepared by Debalaya Sarker, Zhong-Kang Han and Sergey V. Levchenko
Advanced Materials Modeling
Skoltech, May 7, 2020

A quick summary of the exercises

A guideline through the tutorial

This tutorial aims to give a basic introduction to electronic structure calculations for very simple systems. As every DFT code has its own philosophy, this tutorial should also familiarize you with fundamental aspects of using FHI-aims. The goal of the first section is to explain the basic inputs of FHI-aims and to demonstrate that DFT calculations can have predictive power for many observable quantities. The second part introduces geometric optimization of a molecule and how to assess the reliability of the result. Some exercises are marked with a red exclamation mark (!). These exercises demonstrate pitfalls or limitations of the approach.
The practice session consists of three parts:

Part I: Basic electronic structure with FHI-aims

Problem I: Total energy of free atoms
The hydrogen atom

Problem II: Hydrogen Molecule(H_2): bond length !

Part II: Local structure optimization

Problem III: Hydrogen Molecule H_2

As first step please copy the folder `tutorial_2` from `$HandsOn` to your working directory.

Copy `$HandsOn/tutorial_2` to your working directory:

- **cp -r `$HandsOn/tutorial_2` .../Your_Working_directory/**

For every exercise, we also provide solutions and sample input files. They can be found in $HandsOn/tutorial_2/solutions and $HandsOn/tutorial_2/skel/exercise_XX/templates, respectively. However, we strongly recommend to use the provided input files only in case of time shortage. You will maximize your learning progress by trying to generate the input files on your own. In case you get stuck with a particular problem, do not hesitate to ask one of the tutors. For the tutorials, an executable of FHI-aims will be provided on your workstation at `$HandsOn/bin/aims.x`. However, you have to submit the jobs to queue with submission script `$HandsOn/job.sh`

Note: Please do not copy and paste the description in this pdf into your input files. Typically, invisible characters from the formatting are copied, too. FHI-aims will reject these characters and your calculations will not start.

The very basics of FHI-aims

Each calculation should be done in a separate directory containing the two mandatory input files `control.in` and `geometry.in`. FHI-aims is then called in this directory.

In short:

- **Each calculation in a separate directory**
- **2 input files:**
  - `control.in`
  - `geometry.in`
- **copy submission script to working directory**
  - `$HandsOn`/job.sh Path_to_Your_working_directory/
  - **Start calculation**
    `sbatch job.sh`

The above starts a calculation on a single processor, shows the main output on the screen and, at the same time, pipes it to the `output` file. The `output` file contains the basic information and results of the calculation such as the total energy, atomic forces, and so forth. Additional output files might be generated according to the specified settings.
NOTE:

:

- Depending on the job in hand, you can modify the wall time in the submission script `job.sh` in each calculation directory.

```
#!/bin/sh
#SBATCH -p xavier
#SBATCH -J test
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1

module load intel/mkl-11.2.3 mpi/impi-5.0.3
mpirun -np $SLURM_NTASKS $HandsOn/bin/aims.x >& output
```

Figure 1: Sample submission script. You may alter the wall time # SBATCH -t, number of nodes # SBATCH -n depending on the job in hand.

Nuclear positions: geometry.in

The `geometry.in` file contains all information concerning the atomic structure of the system. This includes the nuclear coordinates, which are specified by the keyword `atom`, followed by cartesian coordinates (in units of Å) and the element symbol, called "species" in FHI-aims. In addition, comments can be added using a preceding hash symbol. Fig. 2 shows an example `geometry.in` file for a hydrogen atom.

```
#The hydrogen atom
atom 0.0 0.0 0.0 H
```

Figure 2: An example `geometry.in` file for a hydrogen atom positioned at the origin.

For periodic calculations, lattice vectors can also be given in this file. This will be covered in the next tutorial. In the present tutorial, however, we will stick to non-periodic systems.

Choosing the method: control.in

This file contains all physical and computational settings for the calculation. Fig. 3 shows a minimal example of a `control.in` file, which can be used as a template during the tutorial.

```
#Simplest input settings
################################################

xc  hf
charge 0.0

#####Iteration limit#####################################

sc_iter_limit 300

########Species#########################################
```

Figure 3: Simple physical and computational settings for `control.in`.

In this example, the following options are set:

- `xc`
  This keyword sets the method to be used. For example you can choose the option `hf`, which requests a Hartree-Fock calculation.

- `charge`
  Set the total charge of the system in units of $|e|$. For a neutral system, this is zero.

- `sc_iter_limit`
  Determines the limit of self-consistency steps allowed in the calculation.

Additional remark:
Please note that you also can change the convergence criteria of the self-consistent field (s.c.f) cycle using the keyword `sc_accuracy_rho` for the electron density. If this keyword is not set, FHI-aims uses a default that is rather stringent for most production calculations. The value chosen by the code for a given system can be found in the output file of each run. There are additional criteria that can be required for s.c.f. convergence. However, checking expensive quantities such as forces or stresses directly for s.c.f. convergence can be **very** costly and should be avoided unless there is a definite need.

In addition to these keywords, the `control.in` file must contain the definition of the computational parameters for each species that is specified in `geometry.in`. The order of the species in the listing is irrelevant. FHI-aims is shipped with pre-defined settings for all species which govern the key parameters regarding the numerical accuracy. They include, *inter alia*, the specification of all real-space integration grids, the accuracy of the Hartree potential and, of course, the basis set. For all elements, defaults are provided for three different levels of accuracy: *light*, *tight*, and *really tight*. Additional *intermediate* settings are provided for several much-used elements. The pre-defined settings can be found in the directory

```
$SPECIES_DEFAULTS
```

and should be copied and pasted into `control.in`, e.g. via the command

```
cat $SPECIES_DEFAULTS/really_tight/01_H_default » control.in
```

which pastes the really_tight settings of the H-atom into the `control.in` file. Already the *tight* species_defaults are rather safe and *really tight* settings are overconverged for most purposes. In addition the number of basis functions can be varied, as well as the basis functions themselves. The basis functions associated with a given species are tabulated at the end of these default settings, as shown in Fig. 4

The idea of keeping the species defaults out in the open is that, even if they are not modified, these are the critical accuracy parameters which one might look at to ensure numerical convergence. Each line denotes a specific basis function. They can be read as follows: The first keyword denotes the "type" of the basis function. *hydro* means that this is a hydrogen-like basis function. Some basis functions are of the type *ionic*. They are described in more detail in the manual. The next two symbols correspond to the first two quantum numbers (principal quantum number n, orbital quantum number l) of the basis functions, and the final number corresponds to the "effective nuclear charge" for which this basis function is created. `hydro 1 s 0.85` corresponds to the exact solution for the 1 s basis function of a hydrogen atom if it had a nuclear charge of only 0.85.

The basis functions are classified in "tiers" (i.e., levels of importance). Not all basis functions are enabled by default. Rather, some are commented out using the " # " symbol. They can be included in the calculation by removing the hash symbol from the corresponding lines. Systematically improved calculations can be performed by enabling additional tiers one after another (however, the computational cost for routinely overconverged calculations also increases rapidly).

```
#######################################################################
#
#  FHI-aims code project
#  VB, Fritz-Haber Institut, 2007
#
#  Suggested "safe" defaults for H atom
#  (to be pasted into control.in file)
#
#######################################################################
  species        H
#     global species definitions
    nucleus             1
    mass                1.00794
#
    l_hartree           8
#
    cut_pot             4.0  2.0  1.0
    basis_dep_cutoff    0.d0
#
    radial_base         24 7.0
    radial_multiplier   2
    angular_grids       specified
      division   0.2783  110
      division   0.3822  194
      division   0.5626  302
      division   0.5922  434
      division   0.6227  590
#      division   0.7206  770
#      outer_grid  770
      outer_grid  590
#######################################################################
#
#  Definition of "minimal" basis
#
#######################################################################
#     valence basis states
    valence      1  s   1.
#     ion occupancy
    ion_occ      1  s   0.5
#######################################################################
#
#  Suggested additional basis functions. For production calculations,
#  uncomment them one after another (the most important basis functions
#  are listed first).
#
#  Basis constructed for dimers: 0.5 A, 0.7 A, 1.0 A, 1.5 A, 2.5 A
#
#######################################################################
#  "First tier" - improvements: -1014.90 meV to -62.69 meV
     hydro 2 s 2.1
     hydro 2 p 3.5
#  "Second tier" - improvements: -12.89 meV to -1.83 meV
#     hydro 1 s 0.85
#     hydro 2 p 3.7
#     hydro 2 s 1.2
#     hydro 3 d 7
#  "Third tier" - improvements: -0.25 meV to -0.12 meV
#     hydro 4 f 11.2
#     hydro 3 p 4.8
#     hydro 4 d 9
#     hydro 3 s 3.2
```

Figure 4: Tabulated basis functions for hydrogen. The basis functions are classified in "`tiers`". In this example only the `tier 1` and minimal basis functions are enabled.

Free atoms and simple spin-polarized systems: Additions to control.in

Part I of this tutorial will deal with free atoms and very simple diatomic molecules. For a physically correct treatment, their spin will have to be considered. To do so, modify your `control.in` file as follows:

```
#Sample input file for the calculation of a H atom
################################################

xc  hf
charge 0.0
spin collinear
default_initial_moment 1

#####Iteration limit#####################################

sc_iter_limit 300

########Species#####################################
```

Figure 5: Default physical and computational settings for simple spin-polarized systems in `control.in`.

These basic keywords should be used as default for part I of this tutorial, unless specified otherwise. However, **do not** use these modifications routinely in other calculations if you do not need them – see below for more information:

- `spin`
  This keyword governs the spin treatment. It can be set to `none`, which requests a spin-restricted calculation, or to `collinear`, which requests a spin-unrestricted (polarized) calculation. In a spin-restricted calculation, α and β spins are assumed to be equal. Only one spin-channel is treated and hence, the number of electrons is effectively halved. This accelerates the calculations significantly, typically a factor 2 or more.
  Important: In systems that are safely unpolarized, always use `spin none`.

- `default_initial_moment 1`
  Sets the initial spin of the atoms. The value 1 requests an initial spin moment of 1, i.e., one more spin-up electron than there are spin-down electrons. Only necessary for `spin collinear` calculations.
  Warning: Only ever use the `default_initial_moment` keyword for very simple systems, where all atoms are expected to behave identically. In more complex systems, where some atoms carry spin and others don't, never use `default_initial_moment`. Instead, modify the `geometry.in` file and add individual `initial_moment` keywords to specific atoms there. Initializing the s.c.f. cycle with an unphysical spin state can greatly slow down the calculation and can lead to physically incorrect results.

> **Tip**:
> In principle, one never needs to use `default_initial_moment` at all. It is much simpler (and less error-prone) to place `initial_moment` keywords after any atom to be spin-initialized in `geometry.in` in Figure 6:

```
#The correctly spin-initialized hydrogen atom
atom 0.0 0.0 0.0 H
  initial_moment 1.0
```

Figure 6: `geometry.in` file with spin initialization of a single H atom. Each atom can get its very own initial moment by placing an `initial_moment` tag in the next line for any atom that might carry a spin.

Additional tools and programs

Bash shell:
A short list of the basic bash (command line) commands is given in Appendix II.
All scripts you will need for this tutorial can be found in
`$HandsOn/tutorial_2/utilities`
Visualization tools:
To visualize structures, you may use jmol. A short jmol tutorial video can be found in $HandsOn/tutorial_2/jmol_tutorial.ogv. In case you are not familiar with jmol, use any of your favourite softwares viz. VESTA etc.
Plotting and Editing:
Matplotlib and xmgrace can be used to visualise and plot data. Also, you can use Origin, Gnuplot or any software of your choice that you are already familiar with.

Part I: Basic electronic structure with FHI-aims

Problem I: Total energy of free atoms: The hydrogen atom

In this exercise, we aim to convey the basics of FHI-aims using the hydrogen atom. The hydrogen atom is the simplest non-trivial system possible and the only one for which the exact analytic solution is known. By the end of the first exercise, we will see how various computational methods compare to each other and to the exact solution. From a technical perspective, we will learn how to generate input files, read the standard FHI-aims output, and perform basis set convergence tests.

Getting started - the hydrogen atom

Educational Objectives:

- Become aquainted with running FHI-aims calculations
- Learn how to do systematic basis set convergence

Tasks

1. Generate a simple `geometry.in` file by hand, which contains only a single hydrogen atom, using the example shown in Fig. 2. This corresponds to a single hydrogen atom in a hypothetical ideal gas phase. It is located at the origin of the coordinate system, although its position does not matter here.

2. Generate a simple `control.in` file by hand, using the example `control.in` file given in Fig. 5. Systems with only a single electron can be solved exactly (within the Born-Oppenheimer approximation) using Hartree-Fock. Finally, append the "*really_tight*" species data of H to the end of the `control.in` file, e.g. via the command
`cat $SPECIES_DEFAULTS/really_tight/01_H_default >> control.in`

3. copy `$HandsOn/job.sh` to working directory

4. Now, run FHI-aims:
`sbatch job.sh`

   Once the calculation has finished, open the `output` file. If you find the line "`Self-consistency cycle converged.`" near the end, then your calculation is converged. We are now interested in the total energy. Search for the block

```
| Total energy uncorrected   :   -0.136053823214315E+02   eV
| Total energy corrected     :   -0.136053823214315E+02   eV
| Electronic free energy     :   -0.136053823214315E+02   eV
```

   In this special case, all energies are equal, but this will not be the case if fractionally occupied orbitals were found! For non-metallic systems, as in this tutorial, always use the **Total energy uncorrected** value. (You will learn what the other two values mean in a future lecture.) Compare it with the exact result for the hydrogen atom (0.5 Hartree $\approx$ 13.6057eV).

   TIP:
   In later exercises, to find this value fast and efficiently, use the command
   `grep 'Total energy uncorrected' output`

5. Redo the calculation with different basis sets (`minimal` , `tier1`, `tier2`, `tier3`) by (un)commenting the basis functions at the end of the control.in file. Calculations with minimal basis set are performed by removing all basis functions that are listed in the file. Search the output file to find out how many basis functions are actually used in the calculations. Then, plot the total energy as function of the basis set size. At which tier does total energy converge?

   TIP:
   To plot the results, simply create a text file (e.g., *results.dat*) with two columns, the number of basis functions and the obtained total energy. This file can be plotted directly using xmgrace with the command
   `xmgrace results.dat`

In principle you can use any plotting software of your choice if you dont́ have xmgrace installed viz. Gnuplot, Origin etc.

Method performance

To learn about the performance of difference exchange-correlation functionals, repeat for different methods. Replace `hf` in control.in with

- `pw-lda`
- `pbe`
- `pbe0`

Do all methods converge with basis set size? Do all converge to the same result?

Optional: An optimal basis set

If you browse through the hydrogen basis set, you will note that the hydrogen 1s function is not included. Change that by adding the line

```
hydro 1 s 1
```

at the end of the `control.in` file. Comment out all other basis functions and run the Hartree-Fock calculation again. How close does it get to the exact result?

Problem II: Hydrogen molecule (H_2): bond length and dipole moment

Hydrogen Molecule (H_2)

One of the most influential papers in chemistry for systematic investigation of the performance of DFT was Johnson et al.[1], in which several properties of a large number of diatomic systems were consistently computed and compared to experimental values. In the style of this work, we will calculate the binding curve and atomization energy (ΔH_{at}) for hydrogen molecule (H_2) with two methods.

Eductational Objectives:

- Find the equilibrium bond distance of a simple diatomic molecule.

Tasks:

1. The first task of this exercise will be to find the equilibrium bond distance of H_2 from a series of calculations. Start by creating a `geometry.in.template` file which contains two H atoms, as shown in Fig. 7.

```
#H2 at variable bond distances
atom 0.0 0.0 0.0 H
atom 0.0 0.0 Dist H
```

Figure 7: The geometry data for a H_2 molecule. One H is put on the origin and the other H is located `Dist` Å away from the origin along the z-axis.

   In this example, One H is put on the origin and the other H is located `Dist` Å away from the origin along the z-axis. Hereby, `Dist` is a placeholder which has to be replaced by the actual distance for every calculation.

2. Create a `control.in.template` file, and specify a `hf` calculation for a neutral system. Feel free to copy contents from `control.in` file in the first exercise, but remove the *really tight* species setting for hydrogen and paste the *tight* setting for H into the `control.template` file.

3. Next, run calculations in separate folders for different bond distances (ideally between 0.5Å and 1.2Å with 0.1Å steps, and a denser step width of 0.02 Å between 0.65 Å and 0.85 Å).

For each distance, you should

- create a unique directory
- create the `control.in` and `geometry.in` file from templates
- replace the bond distance place holder `Dist` with the bond distance and
- start FHI-aims.

Run FHI-aims for all bond lengths and plot the total energies vs. the bond length.

Which bond length corresponds to the lowest energy? How does the bond length compare to the experimental bond length of 0.74Å?

4. To compare with experimental values, we compute the atomization energy (ΔH_{at}). In order to calculate ΔH_{at}, we will also need the total energy of the isolated H atom. Use the total energies for the single atom from previous exercise. Check the result carefully

   Calculate the atomization energy (ΔH_{at}) of H_2 by subtracting the free-atom energies from the predicted total energy of H_2 (i.e. the minimum total energy found when varying bond distances).

   $$\Delta H_{at} = E_{tot}^{H_2} - 2E_{atom}^{H} \tag{1}$$

   How does this compare to the experimental value of $\Delta H_{at} = 103.3$kcal mol^{-1} (4.479 eV)?

Optional: Method Performance.

Repeat the bond length determination using `pbe0`. How does the optimal bond length change? How much does the total energy and atomization energy change?

Part II: Local structure optimisation

Problem III: H_2

This exercise covers how to perform geometry optimizations. Specifically, we will relax the H_2 molecule of our previous exercise starting from an initial guess for the geometry and do avoid the equilibrium bond distance.

Educational Objectives

- Learn how to perform a geometry optimization in FHI-aims
- Visualize the relaxation.

1. Fig. 8 shows a `geometry.in` file for starting the relaxation calculationPlease use this as the starting point for your structure relaxation.

```
#H2 at starting bond distance for relaxation
atom 0.0 0.0 0.0 H
atom 0.0 0.0 0.5 H
```

Figure 8: This `geometry.in` file gives a starting point for the molecular H_2.

2. Create a `control.in` file, using the `control.in` file from the previous exercises as a template. For your `xc` functional (i.e. `method`), specify `pw-lda`. This time, use `spin none` Finally, add the keyword `relax_geometry trm 1E-3` to request structural relaxation. Your `control.in` should look similar to the example shown in fig. 9.

```
#This is a sample input file for execise 2 - relaxation calculation of H2
##############################################################################################################

##### Method ######################

  xc pw-lda
  charge 0.0
  spin none

#####  Iteration limit  #####
 sc_iter_limit     100

#####  Relax Initial Geometry ###########
 relax_geometry trm 1E-3
```

Figure 9: The `control.in` file used for the structure relaxation of the cation H_2.

As in the exercises before, the basis set must be included in the `control.in` file. Use the "*tight*" species defaults for H atoms. **FHI-aims control file does not require repeated entries of species defaults.**
`cat $SPECIES_DEFAULTS/light/01_H_default >> control.in`

3. Run FHI-aims.
`sbatch job.sh`

4. To visualize the results, copy the tool `create_relax_movie.py` from `$HandsOn/tutorial_2/utilities` into the working directory. Apply it to the output and pipe the result to a new file using the command
`python create_relax_movie.py output > H2.xyz`
Open the file `H2.xyz` with a visualizer, e.g. jmol.
`jmol H2.xyz`
(You may use any viewer of your choice and can see jmol tutorial video $HandsOn/tutorial_2/jmol_tutorial.ogv)

Important:
Realize that the chosen relaxation criteria of `relax_geometry trm 1E-3` is rather harsh and should only be used for high accuracy and finite-difference calculations.

What does the fully relaxed structure look like?
Do you believe that this could be the actual total energy minimum?
How is the optimized bond length different from the one you have found in the previous problem?

- Check how the final bond length changes with the change of functionals, tiers and relaxation conditions.

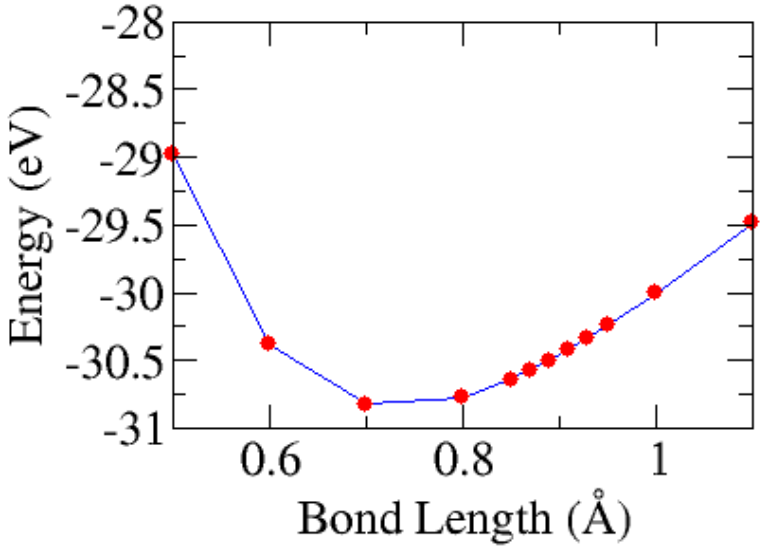


Figure 10: Bond length vs. Total energy plot for a H_2 molecule calculated with hf.

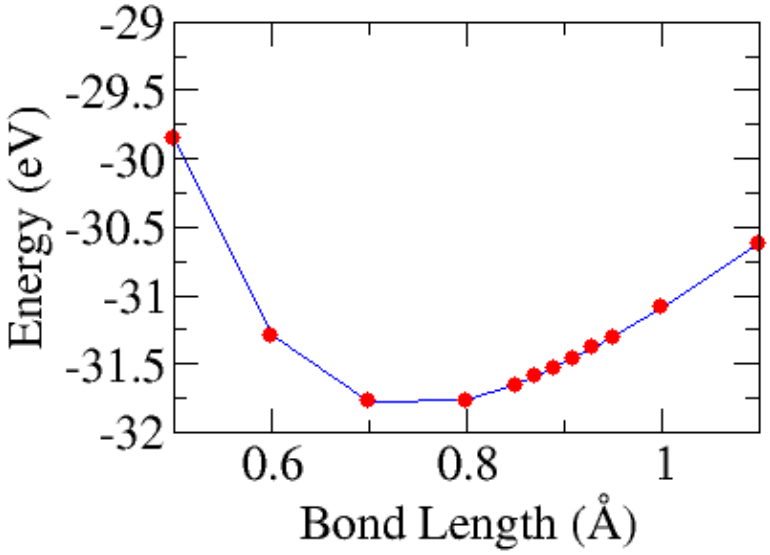


Figure 11: Bond length vs. Total energy plot for a H_2 molecule calculated with pbe0.

Results

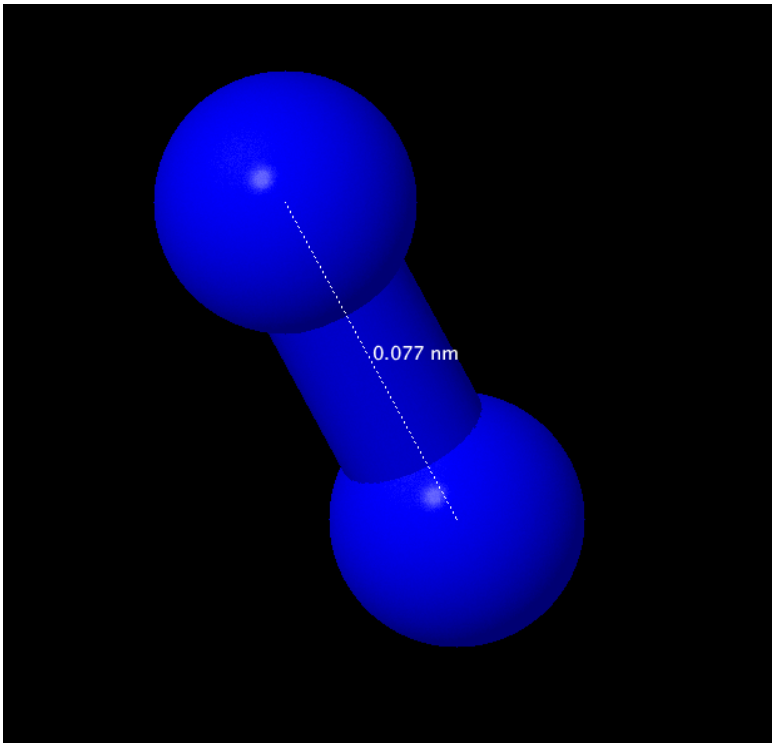


Figure 12: Bond length in H_2 molecule calculated with pw-lda.

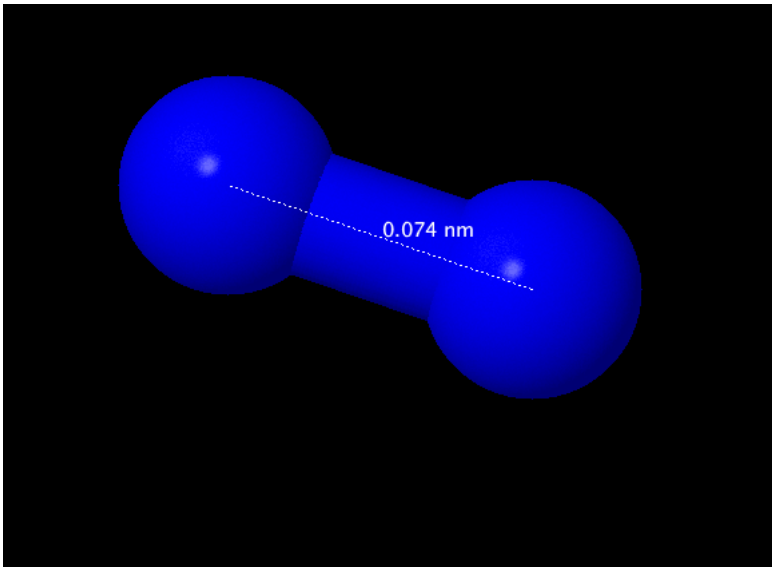


Figure 13: Bond length in H_2 molecule calculated with pbe0.

Appendix

Appendix I: Sample python/bash scripts

Sample script for Exercise 2

Below, you find a sample script in python and bash to calculate a molecule with pre-defined bond distances.

```
#########################################################################
# This is a simple example how to run a series of distances             #
# Note: To run this script, you must be in a directoy which contains    #
#  /  geometry.template file with HF, where the distance between H and F#
#     is determined by a variable named DIST                            #
#                                                                       #
#  /  a control.template file which contains                            #
#     the correct method and basis set specifications.                  #
# Example input files are provided in the same directory as this script #
#########################################################################

import os, shutil
#Variable defintion:
#This variable should point to your local FHI-aims executable
#Variable defintion:
AIMSBIN='aims.x '


f=open('geometry.in.template','r')     #read geometry template
template=f.read()
f.close

#Loop over the distances
for Distance in [0.7, 0.8, 0.85, 0.87, 0.89, 0.9,1 0.93, 0.95,\
                 1.0, 1.1, 1.2, 1.3]:
   print Distance
   # Create directory with the distance as name
   if not os.path.exists(str(Distance)): os.mkdir(str(Distance))
   # copy the control file into the new directoy
   shutil.copy('control.in.template', str(Distance)+'/control.in')
   # the the geometry template  and
   # replace the term DIST by the current distance and
   # copy the new file into the directory
   out=open(str(Distance)+'/geometry.in','w')
   template_out=template.replace('Dist',str(Distance))
   out.write(template_out)
   out.close()
   # Change directory
   os.chdir(str(Distance));
   # Run aims and pipe the output into a file named "output"
   os.system(AIMSBIN+' | tee output')
   # Go back to the original directory
   os.chdir('..')
```

Figure 14: This is a sample python script for calculating a molecule with different pre-defined bond distances.

```
#############################################################################
# This is a simple example how to run a series of distances                 #
# Note: To run this script, you must be in a directoy which contains        #
#  /  geometry.template file with HF, where the distance between H and F#
#     is determined by a variable named DIST                                #
#                                                                           #
#  /  a control.template file which contains                                #
#     the correct method and basis set specifications.                      #
# Example input files are provided in the same directory as this script     #
#############################################################################

#!/bin/bash -l
ulimit -s unlimited

#Variable defintion:
#This variable should point to your local FHI-aims executable
AIMSBIN=aims.x


#Loop over the distances
for Distance in 0.7 0.8 0.85 0.87 0.89 0.91 0.93 0.95 1.0 1.1 1.2 1.3 ;
 do
   echo $Distance
   # Create directory with the distance as name
   mkdir $Distance;
   # copy the control file into the new directoy
   cp control.in.template $Distance/control.in
   # the the geometry template  and
   # replace the term DIST by the current distance and
   # copy the new file into the directory
   sed "s/Dist/$Distance/g" geometry.in.template > $Distance/geometry.in
   # Change directory
   cd $Distance;
   # Run aims and pipe the output into a file named "output"
   $AIMSBIN | tee output;
   # Go back to the original directory
   cd ..
done;
```

Figure 15: This is a sample bash script for calculating a molecule with different pre-defined bond distances.

Sample script for Exercise 10

```
import os, shutil
#Variable defintion:
AIMSBIN='aims.x'
#################################################################
MIXER=['linear', 'pulay']
N_MAX=[3, 5, 8]
PARAM=[0.1, 0.3, 0.2]

print "*****************************************************"
print "EXERCISE X:"
print "*****************************************************"


f=open('control.in.template','r')  #read control template
template=f.read()
f.close

for typ in MIXER:
  if not os.path.exists(typ): os.mkdir(typ)
  os.chdir(typ)

  if typ != 'linear' :
     for num in N_MAX:
        if num == 3:
           para=PARAM[0]
        elif num == 5 :
           para=PARAM[1]
        elif num == 8 :
           para=PARAM[2]
        if not os.path.exists(str(num-para)): os.mkdir(str(num-para))
        os.chdir(str(num-para))
        shutil.copy('../../geometry.in.template', 'geometry.in')
        out=open('control.in','w')     # replace the term TYPE in control
                                       # template by the current mixer
        template_out=template.replace('TYPE',typ)
        # set paramters
        template_out=template_out.replace('PARA',str(para))
        template_out=template_out.replace('NUM',str(num))
        out.write(template_out)# and copy the new file into the directory
        out.close()
        os.system(AIMSBIN+' | tee output') # Run aims and pipe the output
                                           #into a file named "output"
        os.chdir('..')
  else:
     shutil.copy('../geometry.in.template', 'geometry.in')
     out=open('control.in','w')   # replace the term TYPE in control
                                  # template by the current mixer
     template_out=template.replace('TYPE',typ) # set paramters
     template_out=template_out.replace('charge_mix_param',\
                                       '#charge_mix_param')
     template_out=template_out.replace('n_max_pulay','#n_max_pulay')
     out.write(template_out)# and copy the new file into the directory
     out.close()
     os.system(AIMSBIN+' | tee output') # Run Aims
  os.chdir('..')
```

Figure 16: This is a sample python script that tests different settings for the density mixing schemes.

Appendix II: Bash

Bash is a Unix shell and command language for the GNU Project and the default shell on Linux and OS X systems. We will use it to execute most programs and exercises. Below you find a list of the most import commands. It furthermore offers a full programming language (shell script) to automatize tasks e.g. via loops.

Basic control:

```
TAB - auto completion of file or command
Up/Down - See previous commands
CTRL R - reverse search history
Middle Mouse Button - Paste at prompt position
CTRL L - Clear the terminal
!! - repeat last command
```

Basic navigation:

```
ls -a - list all files and folders
ls <folderName> - list files in folder
ls -lh - Detailed list, Human readable
ls -l *.jpg - list jpeg files only
ls -lh <fileName> - Result for file only

cd <folderName> - change directory
cd / - go to root
cd ..- go up one folder, tip: ../../../
pwd - print working directory
```

Basic file operations:

```
cat <fileName> - show content of file
head - from the top
   -n <\#oflines> <fileName>
tail - from the bottom
   -n <\#oflines> <fileName>
mkdir - create new folder
mkdir myStuff ..
mkdir myStuff/pictures/ ..

touch <fileName> - create a file

cp image.jpg newimage.jpg - copy and rename a file
cp image.jpg <folderName>/ - copy to folder
cp image.jpg folder/sameImageNewName.jpg
cp -R stuff otherStuff - copy and rename a folder
cp *.txt stuff/ - copy all of *<file type> to folder

mv file.txt Documents/ - move file to a folder
mv <folderName> <folderName2> - move folder in folder
mv filename.txt filename2.txt - rename file
mv <fileName> stuff/newfileName
  if folder name has spaces use " "
mv <folderName>/ .. - move folder up in hierarchy

rm <fileName> .. - delete file (s)
rm -i <fileName> .. - ask for confirmation
rm -f <fileName> - force deletion of a file
rm -r <foldername>/ - delete folder
```

Extract, sort and filter data:

```
grep <someText> <fileName> - search for text in file
   -i - Doesn't consider uppercase words
```

```
   -I - exclude binary files
grep -r <text> <folderName>/ - search for file names
                               with occurrence of the text
```

Flow redirection -redirecting results of commands:

```
'>' at the end of a command to redirect the result to a file
ex --> ps -ejH > process.txt
'>>' to redirect the result to the end of a file
```

Chain commands

```
'|' at the end of a command to enter another one
    ex --> du | sort -nr | less
```

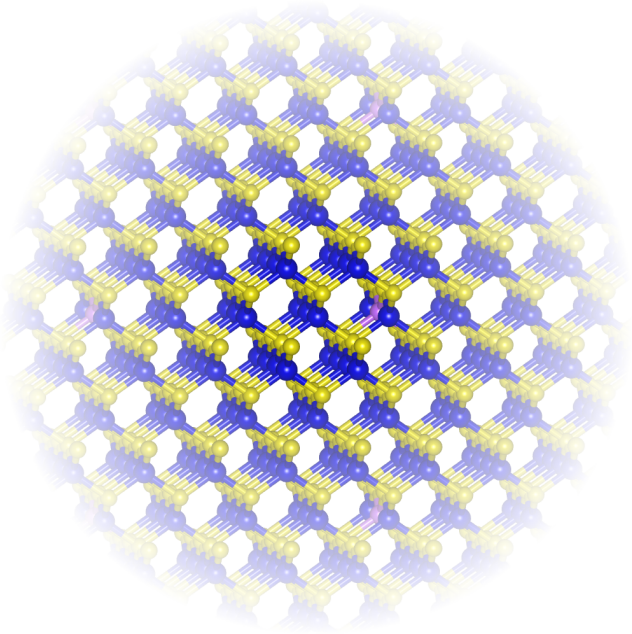
References

[1] B. G. Johnson, P. M. W. Gill and J. A. Pople, *The performance of a family of density functional methods*, The Journal of Chemical Physics **98**, 5612 (1993).

[2] R. S. Mulliken, *Electronic Population Analysis on LCAO-MO Molecular Wave Functions. II. Overlap Populations, Bond Orders and Covalent Bond Energies*, The Journal of Chemical Physics **23**, 1841 (1955).

[3] E. Block, *The Chemistry of Garlic and Onions*, Scientific American **252**, 114 (1985).

FHI-aims Tutorial:
Density-Functional Theory and Beyond
Advanced Materials Modelling



Tutorial III: Periodic Systems
Manuscript for Exercise Problems

Prepared by Debalaya Sarker,
Zhong-Kang Han, Sergey V. Levchenko
CEST, Skoltech

Introduction

This tutorial aims to familiarize you with the basic concepts of periodic density-functional theory (DFT) calculations and with the settings necessary to run FHI-aims. Before we start working on the first problem, a short overview is provided.

The practice session consists of the following parts:

You should work through all the problems to learn about the basic concepts of periodic systems. In the directory `$HandsOn/tutorial_3/`, you can find all the files necessary for this tutorial. Please copy the contents of the `skel/` folder into your own working directory. Dedicated folders have been prepared in the `skel/` directory for each problem. Please use this directory hierarchy, as a few of the directories contain helpful files.

Like before, prepare `control.in` and `geometry.in` files, copy the submission script `$HandsOn/job.sh` to the working directory and starts FHI-aims calculations

If you have difficulty with a particular problem, do not hesitate to ask one of the tutors. In any case, it is perfectly fine to skip the rest of a problem and move on to the next. This also applies if your calculation takes significantly longer than the estimated CPU time for the given problem. Any intermediate results required for later problems are provided in the `reference/` folder. If you like, you may also use this folder to compare to your results.

Take your time to read every task carefully before starting calculations. Each subtask starts with a short summary (grey box) and gives details and hints afterwards. Also, feel free to consult the supplementary information presented in the Appendices.

Part I: Basic properties of solids and convergence tests

In the first Part of this tutorial, we will work on different structural phases of bulk silicon. The correct description of bulk silicon's pressure dependence by Yin and Cohen [1] is one of the early success stories of computational materials science. In this Part, we show how to calculate basic properties of solids such as lattice constants, band structures, and density of states.

Please use the basic settings given in Fig. 1 as default for Part I of this tutorial (unless specified otherwise).

```
# Physical settings
  xc                 pw-lda
  spin               none
  relativistic       atomic_zora scalar

# k-grid settings  (to be adjusted)
  k_grid  nkx nky nkz
```

Figure 1: Default physical and computational settings for `control.in` for Part I. This file can be found in `skel/problem_1/control_part1.in`.

The Perdew-Wang LDA (`xc pw-lda`) exchange-correlation functional will be used for all calculations. The effect of using different xc functionals has been discussed in "Tutorial 2: Basics of Electronic-Structure Theory". Silicon is known to be non-magnetic, so no explicit spin treatment is needed. The "`relativistic atomic_zora scalar`" setting is not strictly necessary for silicon, since the nuclear charge of silicon ($Z = 14$) is still small enough to allow for a non-relativistic treatment. But as the correction is computationally inexpensive, it does not hurt to use it, either. However, never compare total energies from different relativistic settings, as they will differ.

The SCF settings section of `control.in` has already been discussed in detail in "Tutorial 2: Basics of Electronic-Structure Theory". The `k_grid` setting section will be discussed in this tutorial.

For the species settings, please use the default "light" species settings for Si in:
`$SPECIES_DEFAULTS/light/14_Si_default`.

Problem I: Generation and visualization of bulk structures

Our first step towards studying periodic systems with FHI-aims is to construct periodic geometries in the FHI-aims geometry input format (`geometry.in`) and visualize them. After that, we will learn how to set basic parameters in `control.in` for periodic calculations. Finally, we will compare total energies of different Si bulk geometries.

Setting up and visualizing `geometry.in`

- Construct `geometry.in` files for the Si *fcc*, *bcc*, and diamond structures. Use the approximate lattice constants a of 3.8 Å for *fcc*, 3.1 Å for *bcc*, and 5.4 Å for the diamond structure.
- Visualize them (e. g. with Jmol).
- Do not perform any calculations yet.

To set up a periodic structure in FHI-aims, all three lattice vectors as well as the atomic positions in the unit cell must be specified. The lattice vectors are specified by the keyword `lattice_vector`. There are two ways to specify the atomic positions. As in the cluster/non-periodic case, you can specify absolute Cartesian positions with the keyword `atom`. Alternatively, you can specify the atomic positions in the basis of the lattice vectors, the **fractional** (or commonly, **direct**) coordinates, with the keyword `atom_frac`. The fractional coordinates s_i are dimensionless and the coefficients for the linear combination of the lattice vectors $\mathbf{a}_i$. Written out as a formula, this linear combination reads as follows

$$\mathbf{R} = s_1 \cdot \mathbf{a}_1 + s_2 \cdot \mathbf{a}_2 + s_3 \cdot \mathbf{a}_3, \tag{1}$$

where $\mathbf{R}$ is the Cartesian position of the specified atom.

For example, *fcc* Si with a lattice constant $a = 4$ Å is defined by

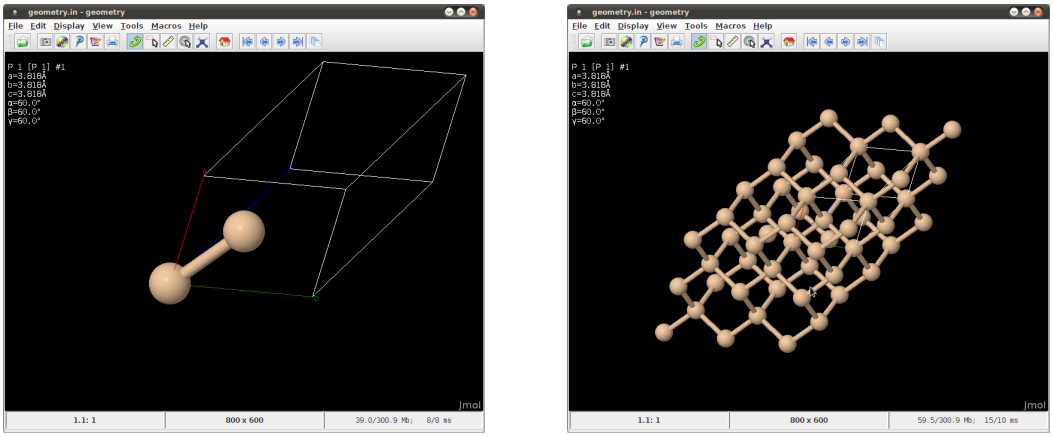


Figure 2: Single unit cell (left) and periodic images (right) of diamond Si in Jmol.

```
lattice_vector   0.0   2.0   2.0
lattice_vector   2.0   0.0   2.0
lattice_vector   2.0   2.0   0.0
atom_frac        0.0   0.0   0.0   Si
```

A full set of lattice vectors and atomic positions of primitive unit cells (the unit cell with the smallest volume, containing the bare minimum number of atoms necessary to replicate the system when all periodic images are included) for *fcc*, *bcc*, and diamond can be found in Appendix I. In Fig. 20 in Appendix I the simple cubic and primitive unit cells of *fcc*, *bcc*, and diamond are shown. Atomic positions are provided in Cartesian coordinates. Please note that the values in `geometry.in` must be provided explicitly and not in terms of the formulas presented in Appendix I; specifying a value of `4.0/2.0` instead of `2.0` will cause your calculation to fail.

The simplest way to check the `geometry.in` file is to visualize the corresponding geometry. This should **always** be done before any calculation, to verify that the structure is plausible (and that no atoms are extra or missing). For periodic structures in FHI-aims, we recommend Jmol, an open-source Java viewer for chemical structures in 3D. Information on the program and the source code can be obtained from `http://jmol.sourceforge.net`. To visualize a structure given in `geometry.in` with Jmol (Fig. 2), please type

```
jmol geometry.in &
```

To get periodic images, click with the right mouse button inside the Jmol drawing area and choose "Symmetry" → "Reload {444 666 1}".

Setting up `control.in` and running FHI-aims

- Prepare a `control.in` file using 3×3×3 k-points and the settings given in the introduction of part I (see Fig. 1). These settings can be found in `skel/problem_1/control_part1.in`.
- Calculate total energies of each of the different phases as a function of lattice constant a. For this, consider 7 different values of a in steps 0.1 Å around the lattice constants given above for each structure.

[Estimated total CPU time: 3 min]

The `control.in` for periodic calculations looks much the same as for the cluster case, as the underlying numerics are the same. There is one important difference, though: a k-grid for the Brillouin zone integrations must be specified. For example, to specify a 3×3×3 k-grid, the following line must be included in `control.in`:

```
k_grid 3 3 3
```

The k-grid points are defined in terms of the reciprocal lattice vectors which are generated from the real-space lattice vectors as defined in `geometry.in`. The ordering of real-space lattice vectors in `geometry.in`

determines the ordering of reciprocal lattice vectors in the code. For systems with real-space lattice vectors that are not equivalent by symmetry (such as a surface calculation where one lattice vector is much longer than other lattice vectors), the ordering will matter. You will see that for a small unit cell this k-point density is *never* enough. But we can try to find out what happens. In the next problem, the k-point settings will be discussed in detail.

In this tutorial, we call FHI-aims with the command similar to the one we used in "Tutorial 2: Basics of Electronic-Structure Theory". In order to run the calculation on the 4 available (physical) processorsiIt is recommended that you modify now the `job.sh` submission script as Fig 3 to run calculations with the number of physical processors (here 4).

```
#!/bin/sh
#SBATCH -p xavier
#SBATCH -J test
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 4

module load intel/mkl-11.2.3 mpi/impi-5.0.3
mpirun -np $SLURM_NTASKS $HandsOn/bin/aims.x >& output
```

Figure 3: Sample submission script. You may alter the wall time # SBATCH -t, number of nodes # SBATCH -n depending on the job in hand.

Next, in your workstation, use the following command:

```
sbatch job.sh
```

Make sure you use MPI, otherwise your calculations will take (roughly) 4 times longer. The estimated CPU time given with the exercises refers to the calculations with 4 processors. It is good practice to use a separate directory for every run of FHI-aims in order to preserve the exact input files along with the output files.

In this exercise, we compare energies of different lattice structures as a function of lattice constant. Each calculation can be prepared and started by hand, in principle, but we strongly suggest to use python/bash scripts

Now, extracts the total energies and write them to the file `energies.dat`, along with the lattice constants. For the next subtask, it is advantageous to write out the total energy per atom, not per unit cell, which makes a difference for the diamond structure. (In the case of fcc and bcc, each unit cell contains one atom, so the total energy per atom and the total energy per unit cell are the same quantity.)

Plotting total energies

- Plot the total energy per atom of each structure as a function of the lattice constant (e.g. with xmgrace).
- What is the most stable bulk phase of Si according to your results?

After running the code, plot your data (given in `fcc/energies.dat`, `bcc/energies.dat`, and `diamond/energies.dat`) using for example xmgrace by typing:

```
xmgrace -legend load \
  fcc/energies.dat bcc/energies.dat diamond/energies.dat &
```

All curves should have minima (if not, check the list of lattice parameters) and similar energy ranges (if not, make sure that you properly printed out the total energy per atom, and not the total energy per unit cell).

You should see that, with the current computational settings, the diamond Si phase is unfavorable compared to the other two phases by about 0.1 eV. However, the experimentally most stable phase is the diamond structure. We will show in the next two problems that the too coarse $3\times3\times3$ k-grid is the reason for this disagreement.

Problem II: Energy convergence tests

The results of the last problem were not quite physical. As will be shown later, this is because the calculations were not converged. Here, we will explicitly check total energy convergence with respect to the k-grid and to the basis set.

Convergence with k-grid

- Calculate the total energies for **ALL** the Si phases of Problem I as a function of the lattice constant for k-grids of 8×8×8, 12×12×12, and 16×16×16. Otherwise, use the same computational settings (`control.in`) and the same lattice constants as in Problem I.
- Prepare a plot with all total energies drawn against lattice constant using xmgrace, as you did in Problem I. Add the 3×3×3 results from Problem I, too.
- Which k-grid should be used to achieve convergence within 10 meV?

[Estimated total CPU time: 6 min]

You should dedicate one directory for every series of these calculations. These calculations should be done exactly as in Problem I but with the appropriate changes to `control.in`.

In the metallic *fcc* and *bcc* phases, the total energy of the 3×3×3 calculation is about 0.3 eV lower than the most accurate (16×16×16) calculation. The larger part of this error is already fixed by the 8×8×8 k-grid, which is still off by about 30 meV. The 12×12×12 grid, on the other hand, is converged within about 5 meV. For the semiconducting diamond Si phase, the total energy of the 3×3×3 calculation is about 0.3 eV too high, but the convergence is already very good for an 8×8×8 k-grid.

- **In general, metals (like Si *fcc* & *bcc*) or small cells require a denser k-grid compared to semiconductors (Si diamond) or large cells.**

In conclusion, we can use a 12×12×12 k-grid for *fcc* and *bcc* Si as a good compromise of high accuracy and reasonable computational time. For simplicity, we use the same grid also for diamond Si although 8×8×8 would be enough in that case.

Convergence with basis set size

- Calculate the total energies for **your** phase of Si as a function of the lattice constant for the `minimal` and the `tier1` basis sets. Use the same lattice constants and computational settings as in Problem I together with the 12×12×12 k-grid.
- Again, prepare a plot with the total energies. Add the results for the `minimal+spd` basis set (the default for the "light" species settings) from the k-point convergence test above.

[Estimated total CPU time: 4 min]

Use your `control.in` file from the previous problem for the 12×12×12 k-point grid. In order to change the basis size settings, you should have a look into the species-dependent settings within `control.in`. There, you will find a line starting with " `# "First tier" - ...` ". Each line after this defines a basis function which is added to the `minimal` basis. Right now, there is one additional function for each valence function (s and p) as well as a d function to allow the atoms to polarize. This is what we call `minimal+spd` in the context of this tutorial. In quantum chemistry and in particular the Gaussian community, this type of basis set is often called "double zeta (ζ) plus polarization" (DZP).

To run FHI-aims with a `minimal` basis, simply comment out these three lines by prepending a "`#`" character. To run FHI-aims with a full `tier1` basis set, uncomment all four lines following the statement " `# "First tier" - ...` " by removing the initial "`#`" character.

You can see that the `minimal` basis gives completely unphysical results; the energetic minimum is strongly shifted towards larger lattice constants. The `minimal` basis lacks the flexibility to give reasonable geometries. On the other hand, the binding curve does not change drastically from `minimal+spd` to the

full `tier1` basis set whereas the computational effort increases significantly by adding the f functions from `minimal+spd` to full `tier1`, as there are seven such functions (l=-3,-2,...,2,3). You could also check `tier2`. In that case, though, it would be worth using better integration grids (tighter) etc. as well.

While the total energy difference of about 60 meV between `minimal+spd` and `tier1` is still larger than what we were aiming for in the case of the k-grid, we can make use of the fact that the total energy is variational so that a large part of the basis set error actually cancels nicely in energy differences between bounded structures.

Bonus: Effect of Gaussian broadening of the Kohn-Sham occupation numbers

Bonus: Please skip this subtask if you run out of time or you are inexperienced in modifying scripts.

- Calculate the total energies for *fcc* Si as a function of the lattice constant for a Gaussian broadening of $\sigma = 0.1\,\text{eV}$. Use the same lattice constants and computational settings as before with a 12×12×12 k-grid and the `minimal+spd` basis.
- Prepare a plot with the corrected, uncorrected total energies, and the electronic "free energies" for a broadening of $\sigma = 0.1\,\text{eV}$ and the default value of $\sigma = 0.01\,\text{eV}$ from the previous calculations.

[Estimated total CPU time: 2 min]

Use your `control.in` file from the previous problem, with a 12×12×12 k-point grid and the `minimal+spd` basis. You can explicitly set the Gaussian broadening to $\sigma = 0.1\,\text{eV}$ by specifying

```
occupation_type gaussian 0.1
```

in `control.in`.

FHI-aims always outputs three different energies. While these energies are all the same for systems with a gap, they differ for metallic systems with finite Gaussian broadening. The "`Total energy uncorrected`" gives the value of the Kohn-Sham energy functional for the final self-consistent electronic structure. However, due to the Gaussian broadening, the self-consistency procedure does not minimize this total energy but a "free energy" specified right of "`Electronic free energy`". From these two numbers, FHI-aims back-extrapolates to the total energy without broadening and writes the resulting number of "`Total energy corrected`". For true metals, it is generally best to make use of the correction. For finite systems and in particular for isolated atoms, however, the back-extrapolation is unphysical and should not be used.

For diamond Si, the Gaussian broadening makes no difference at all as long as the broadening σ is small compared to the band gap. The first thing to notice for the metallic phases is that all of these numbers agree with each other within about 2 meV. For the default broadening of $\sigma = 0.01\,\text{eV}$, the energies even agree within 0.1 meV. It can be shown using the variational principle that the total energy always increases and the electronic "free energy" always decreases for finite broadening.

For the following calculations, we will use the default value of $\sigma = 0.01\,\text{eV}$ because there is no benefit in convergence by increasing this parameter for the studied phases of Si. We will stick to the corrected total energy for the periodic systems in this part of the tutorial as it is the most accurate value for metals and makes no difference for semiconductors.

Problem III: Unit cell relaxation

You have seen in "Tutorial 2: Basics of Electronic-Structure Theory" how to optimize the positions of atoms in cluster systems. To obtain optimized periodic structures, both the atomic positions and the lattice vectors must be optimized. This will be the topic of this problem.

- Fully relax (both atomic coordinates and lattice vectors) a distorted *bcc* Si structure (see below) with the computational settings you have used before for the Si crystals.
- Analyze the resulting structure (lengths of lattices vectors, angles between them, and atomic positions) and compare it to the of diamond Si obtained from experiments.

[Estimated total CPU time: 10 min]

The basic idea is to perform a structure relaxation from *bcc* to diamond Si. When performing a local structure optimization, we try not to go uphill in the energy landscape or cross energy barriers. Thus, there should be a path from the starting structure to the desired local minimum with very few barriers in between. However, this means that we have no assurance that we have found the global minimum, as this may lie over a notable energy barrier from our starting geometry. In order to cross such energy barriers, one can do, for example, variable cell-shape molecular dynamics, but this is out of the scope of this tutorial.

Since we desire a diamond structure, we must start with at least 2 atoms, as it is crucial that the starting unit cell be compatible with the primitive cell (i.e. contain integer multiples of the number of atoms in the primitive cell). Hence, our starting unit cell is a cubic *bcc* structure consisting of 2 atoms. Of course, we cannot start with the ideal cubic *bcc* structure because we will be stuck in the local minimum for this highly-symmetric structure. Even if we distort the starting geometry significantly (atomic positions & lattice vectors), we can easily end up in an unwanted local minimum. Therefore, we will provide a suitable starting geometry, but feel free to try other initial geometries when you are finished with the given one. You can find this geometry in the directory `skel/problem_4/`:

```
lattice_vector   3.1   0.4   0.4
lattice_vector   0.45  3.1   0.4
lattice_vector   0.45  0.45  3.1
atom_frac        0.0   0.0   0.0  Si
atom_frac        0.3   0.3   0.3  Si
```

To perform a full optimization of the crystal structure, add three lines to your `control.in` from the previous problem (k-grid 12×12×12, minimal+spd basis set). The line

```
relax_geometry trm 1E-2
```

requests a structure relaxation for atoms until residual forces on the atoms smaller than 10^{-2} eV/Å have been achieved, and the line

```
relax_unit_cell full
```

enables full optimization of the lattice vectors (lattice vector lengths and the angles between them).

The structure optimization with the provided starting geometry should take 12 relaxation steps, each relaxation step containing an SCF cycle for the geometry predicted for that step. If you take a closer look into the FHI-aims output, you will see that not only are the atomic forces calculated but also a quantity called the "stress tensor". In essence, the stress tensor is a measure of the forces acting on the unit cell itself. The final relaxed geometry will be written to `geometry.in.next_step`.

You can visualize the geometries along the relaxation path if you run the script `$AIMSUTILS/create_geometry_zip.pl` (one of the utilities distributed with FHI-aims) and specify the main FHI-aims output file for your relaxation as an argument for this script. Unzip the resulting file `geometries.zip` if you want to see the individual geometries, or use Jmol and the files generated by `$AIMSUTILS/create_geometry_zip.pl` to watch a short movie of the relaxation by typing the command "`jmol -s geometries.spt`".

The resulting geometry is a primitive diamond structure. The angles between the lattice vectors are about 60°, and the length of the lattice vectors matches approximately the result from the Murnaghan fit in the previous problem. The vector connecting the two atoms is $(0.25, 0.25, 0.25)$ in fractional coordinates, as is expected for the diamond structure. Increasing the basis set size also improves the resulting geometry. Ideally, one should start a relaxation calculation with FHI-aims using "light" settings. Once the light relaxation has converged, the resulting geometry (e.g. from the file `geometry.in.next_step`) is used as an input for a calculation with "tighter" settings. This saves a significant amount of time, as the relaxation with (relatively) inexpensive "light" settings will draw the system into the neighborhood of the local minimum, allowing the tighter relaxation to focus on "finishing the job" of detailed convergence of the geometry.

Problem IV: Electronic band structure & density of states

The electronic band structure describes energy levels of electrons inside a solid. It gives information about ranges of energy that electrons can occupy (the bands) and the ranges of energies where no electrons may be found (the band gaps). Many properties of a solid can be deduced from its band structure, e.g., if it is a metal, semiconductor, or insulator.

- Calculate the electronic structure of diamond Si using the equilibrium geometry found in Problem III.
- Calculate the band structure along the high symmetry lines
  $L \to \Gamma \to X \to W \to K$.
- Calculate the density of states (DOS) using an energy range of -18 eV to 0 eV, Gaussian broadening of 0.1 eV, a k-grid of $12\times12\times12$, and `dos_kgrid_factors` of 5 for each k-grid direction.

[Estimated total CPU time: 3 min]

Use your `control.in` and `geometry.in` for the equilibrium lattice parameter for diamond Si from Problem III. To calculate the band structure, the high symmetry k-points in reciprocal space must be supplied to FHI-aims. An example excerpt from `control.in` corresponding to the first part of the suggested path, with 21 points per path[1], is:

```
# diamond band structure:
output band  0.5   0.5   0.5    0.0   0.0   0.0    21    L  Gamma
output band  0.0   0.0   0.0    0.0   0.5   0.5    21    Gamma  X
output band  ...
```

In each line, the first three numbers are the coordinates of the starting point in units of the reciprocal lattice vectors. The next three numbers specify the end point. The band structure is then calculated along the path connecting these two points.

The last two entries are not mandatory for the calculation inside FHI-aims, but they provide the label of the specified k-points for the plotting script which you are going to use after the calculation. Please refer to Appendix II for the location of the other high symmetry k-points in the Brillouin zone for the materials in this tutorial.

Despite the ubiquity of band structure calculations in condensed matter physics and materials science, a systematic list of high-symmetry k-paths for all possible Brillouin zones (i.e. all possible materials) was proposed only recently in 2010 by Curtarolo and Setyawan in Ref. [5] for usage in the AFLOW materials database (website `http://aflowlib.org`). For comparison, the original Bloch theorem underlying the band structure formalism was proposed in 1928.

The density of states (DOS)

The density of states is one of the basic concepts in solid state physics. The DOS around the Fermi level is of particular interest as it is one of the fundamental quantities for a material, specifying whether a material is conducting, semiconducting, or insulating. Many material properties depend on the DOS, notably the electrical and thermal conductivity.

The number of states n within a given energy interval $(\epsilon_0 - \Delta\epsilon) < \epsilon < (\epsilon_0 + \Delta\epsilon)$ per unit volume V_{cell} is given by

$$n = \int_{\epsilon_0-\Delta\epsilon}^{\epsilon_0+\Delta\epsilon} \mathrm{d}\epsilon\, g(\epsilon) \tag{2}$$

where $g(\epsilon)$ is the density of states (DOS). In a free atom or an isolated molecule, the DOS consists of a series of discrete energy levels (δ peaks) and can be written as

$$g(\epsilon) = \sum_i \delta(\epsilon_i - \epsilon). \tag{3}$$

In a periodic system, the single particle energies become k dependent and the DOS continuous. The number of states per energy is averaged over k

$$g(\epsilon) = \frac{1}{V_{\text{BZ}}} \sum_i \int_{\text{BZ}} \mathrm{d}^3k\ \delta(\epsilon_{i,k} - \epsilon). \tag{4}$$

In order to calculate the density of states numerically, we have to replace the integral over the Brillouin zone (BZ) in Eq. (4) by a sum over k-points. In the case of infinite k-points, this replacement is exact.

[1] We choose 21 points per path, instead of 20, as the endpoints are included in the number of k-points. This value breaks the k-path into 20 evenly spaced intervals.

However, to compensate the deficiency of a finite grid, we broaden the $\delta(\epsilon_{k,i} - \epsilon)$ distribution by a Gaussian function with a broadening factor σ,

$$g(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \frac{1}{n_k} \sum_i \sum_k \exp\left[-\frac{1}{2} \left(\frac{\epsilon - \epsilon_{k,i}}{\sigma} \right)^2 \right]. \tag{5}$$

To output the DOS in FHI-aims, the following lines are added to `control.in`:

```
output dos -18. 0. 200  0.1
```

The first two values define the energy window where the DOS should be calculated: the first value is the lower energy bound and the second value is the upper energy bound. The third value is an integer specifying how many points to include in the energy window, and the last value is the Gaussian broadening σ used in Equation 5. All energies (both bounds and broadening) are given in eV.

Small changes in the shape of a peak in the DOS will negligibly affect the total energy. Therefore, a rather coarse k-grid (defined by the keyword `k_grid`) in combination with rather broad choices of σ (given by "`occupation_type gaussian`") can be used for the SCF cycle. However, to resolve fine features in the DOS, a denser k-grid to include more terms in the the sum over k-points in Eq. (5) is necessary. After self-consistency is reached, the DOS can be computed using an interpolated k-grid which is made denser by factors n_1, n_2, n_3, respectively. The factors n_1 to n_3 are given in `control.in` with the keyword:

```
dos_kgrid_factors 5 5 5
```

The density of states is calculated on a denser grid after the SCF cycle. The dimensions of the new k-point grid are $k_1 \times n_1$, $k_2 \times n_2$, $k_3 \times n_3$, where k_i are dimensions of the old k-point grid.

Note that two DOS's will be output into separate files: KS_DOS_total_raw.dat, where the zero of energy has not been altered, and KS_DOS_total.dat, where the zero of energy has been shifted to the computed Fermi level.

In order to visualize the band structure, some post-processing is needed after the FHI-aims run. This may be done by the script `aimsplot.py` (provided in `skel/problem_5/`) as long as the output, `geometry.in`, and `control.in` files are in the same directory. Simply run " `python aimsplot.py` " without any arguments in this directory. **NOTE: perform this task on your on laptop/computer as the server does not have all the required python modules.**

Alternatively, copy the perl script `aims_band_plotting.pl` from path `$HandsOn/tutorial_3/skel/problem_5/` to your working directory. Run `perl aims_band_plotting.pl` Then, plot the `band_structure.dat` file with xmgrace or other plotting software.

You see a band structure with an indirect band gap of about 0.5 eV. Please note that the energy zero is at the Fermi energy calculated by the code (which for an insulating system may lie anywhere within the band gap) and not at the valence band maximum, i.e. the highest occupied energy level for an insulating material The resulting LDA band gap is much smaller than the experimental band gap of silicon (1.17 eV [2]). This disagreement is commonly called the "band gap problem" of (semi-)local functions.

Part II: Dealing with a crystal surface in aims

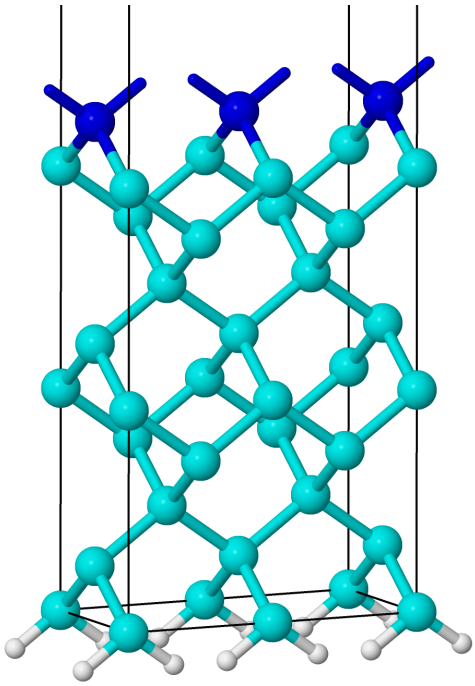


Figure 4: The hydrogen saturated ideal 2×1 Si(001) surface. The cyan (light) atoms correspond to the bulk Si atoms, the blue (dark) atoms are the surface Si atoms. The white atoms on the bottom layer are hydrogen atoms.

In the second part of this tutorial, we demonstrate standard techniques used to describe surfaces. We will model the clean Si(001) surface, which is one of the most technologically relevant surfaces. The surface reconstruction at low temperatures (driven by dangling silicon bonds at the surface) was unclear for many years. After much debate in the literature arising from differing models used to describe different feature observed in experimental (LEED, STM) and theoretical approaches, it has been shown by direct evidence in STM by Wolkow [6] that the main surface feature at low temperatures is an asymmetric Si dimer in a 2×1 reconstructed surface unit cell.

Please use the settings given in Fig. 5 of Part II of this tutorial together with the "light" species defaults for silicon. Most of these settings were also used in Part I of this tutorial, thus you can take your `control.in` from Part I, making sure to adjust k-grid settings. Saturating the dangling Si bonds on the bottom layer of the slab will require the addition of hydrogen to the system, so we additionally need to add the "light" species defaults for hydrogen, which you can find in `$SPECIES_DEFAULTS/light/01_H_default`, to our `control.in` file. As may be observed in Fig. 4, each silicon atom is to be saturated by two hydrogen atoms.

```
# Physical settings
  xc                  pw-lda
  spin                none
  relativistic        atomic_zora scalar

# k-grid settings
  k_grid    12   24   1
```

Figure 5: Physical and computational settings for `control.in` for part II. This file can be found in `skel/problem_6/control_part2.in`.

Problem V: Electronic structure of crystal surfaces

The geometry of four layers of Si(001)-(1×1) in diamond structure is given in Appendix I. Please note that by convention the surface is rotated with respect to the bulk structure by 45° around the z axis.

Please use the optimized lattice constant for diamond Si obtained in Part I.

A slab has two surfaces (by definition), a top and a bottom surface (see Fig. 4). However, in most cases the system of physical interest is essentially an isolated surface, that is, there are enough "bulk" layers between the two surfaces of the slab that they are essentially isolated from one another and are independent experimentally. Here we are interested in the electronic properties of the top surface and would like to minimize the impact of the bottom surface on our calculation. To prevent physical states on the bottom surface (arising from the dangling Si bonds) from appearing in the fundamental gap, the bottom silicon surface is saturated with hydrogen atoms in such a way as to mimic the bonding characteristics of bulk silicon. Each silicon atom in the bottom surface needs two hydrogen atoms placed at a distance of 1.5 Å in the direction where the next set of silicon atoms would have been in the bulk geometry. One can argue that the atomic environment of the hydrogen-saturated silicon atoms will then mimic the atomic environment of bulk silicon.

For your convenience the `geometry.in` file for the ideal hydrogen-saturated four layer slab is provided in:

$HandsOn/tutorial_3/skel/problem_6/01_ideal_2×1_4_layers

One will need a sufficient amount of vacuum between surfaces so that the surfaces do not interact through the vacuum. In many DFT codes (especially codes implementing plane-wave basis sets), you would need to run several calculations with different vacuum thicknesses to find the smallest value which gives physical results. This is because, for many DFT codes, increasing the size of the computational cell substantially increases the basis set size and thus the runtime of a calculation, even if no additional atoms are added to the system. However, in codes implementing localized basis functions (such as FHI-aims), there is negligible computational cost in adding additional vacuum to the system[2], so you may easily use an relatively large vacuum thickness of $L_{\text{vac}} = 100$ Å or more from the onset without a noticeable performance impact.

Band structure and DOS calculation

- Prepare a `control.in` according to the specified settings.
- Calculate the density of states (DOS) and the surface band structures for 4 layer slab along $\bar{\Gamma} \to \bar{J} \to \bar{K}$ (see Appendix II).

[Estimated total CPU time: 3+13 min]

There are two important issues to note for the optimal k-grid for this system. First, there should be no interaction between different periodic images of the slab in z direction. Therefore, only one k-point is needed on that axis. (If you did need more than one k-point along that axis to yield converged results, this would imply that periodic images in that direction were interacting.) Second, the lattice vector in x direction is twice as large as the lattice vector in y direction. As this gives a shorter periodicity in k_y direction, the number of k-points in the first direction can be half of that in the second direction. We use a well converged k-grid of $12 \times 24 \times 1$ (see Fig. 6).

Similar to Part I, for calculating the band structure and DOS, add the following lines in the `control.in` file:

```
# Si 2x1 surface band structure:
output band  0.0   0.0   0.0    0.5   0.0   0.0    21    Gamma  J
output band  0.5   0.0   0.0    0.5   0.5   0.0    21    J  K
```

and for the DOS

```
output dos -18. 0. 200  0.1
dos_kgrid_factors 5 5 1
```

To visualize the band structure, use the script `aimsplot.py` Simply run the script in the directory that contains the input and output files of FHI-aims.

In order to perform structure relaxation such that the residual forces on atoms are smaller than 10^{-2} eV/Å, add the following line to `control.in`:

```
relax_geometry trm 1E-2
```

[2] In fact, this is one of the main strengths of localized basis functions.

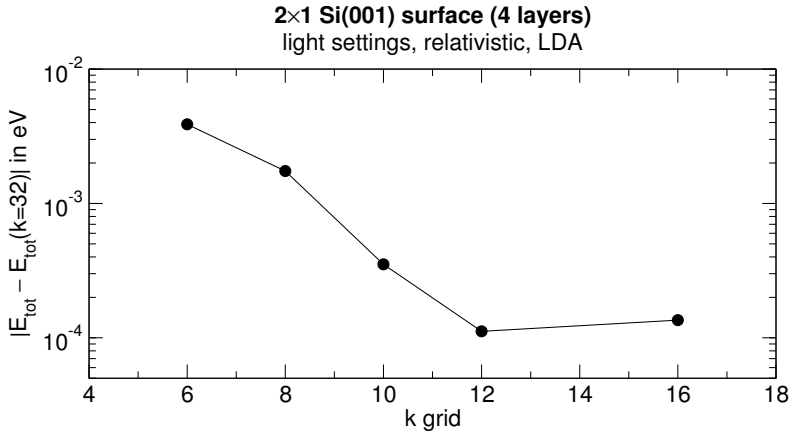


Figure 6: Convergence test for the k-grid of a 4 layer 2×1 Si(001) surface.

During a structure optimization, commonly only the surface-like parts of the structure are allowed to relax, and the bulk-like parts are kept fixed. This is done by the keyword `constrain_relaxation`, which fixes the position of the previously specified atom in `geometry.in`. In the FHI-aims manual, you may find different options, but for constraining all coordinates of an atom, use the flag "`.true.`". In the `geometry.in` file, please write the following line right under the atom which should be kept fixed:

```
constrain_relaxation .true.
```

An example excerpt from `geometry.in` reads like this:

```
atom -1.2063524529754976  0.0000000000000000  -0.8530200000000001  H
  constrain_relaxation .true.
atom  1.2063524529754976  0.0000000000000000  -0.8530200000000001  H
  constrain_relaxation .true.
atom ...
```

For the sake of time, here constrain all atoms except the top layer (i.e. top two Si atoms). For publication-quality calculations, more surface layers should be relaxed.

- Perform k-point convergence, basis set convergence for surface.

[Estimated total CPU time: 3+13 min]

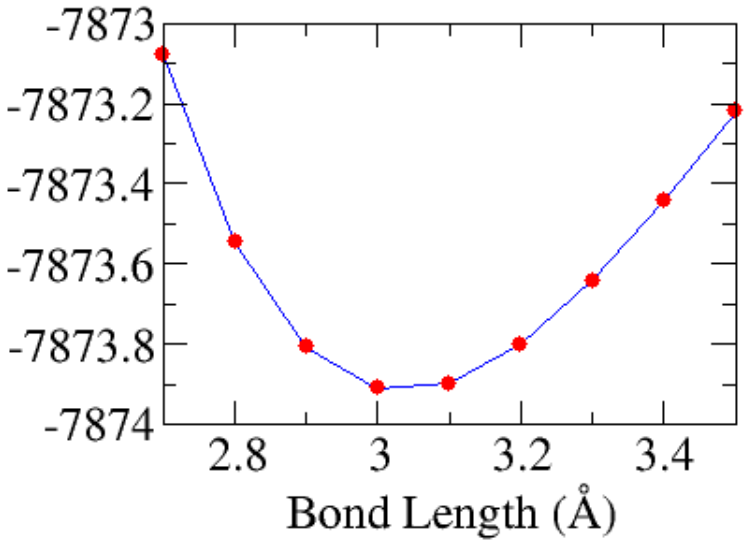


Figure 7: Bond length vs Total energy in bcc Si calculated with pw-lda.

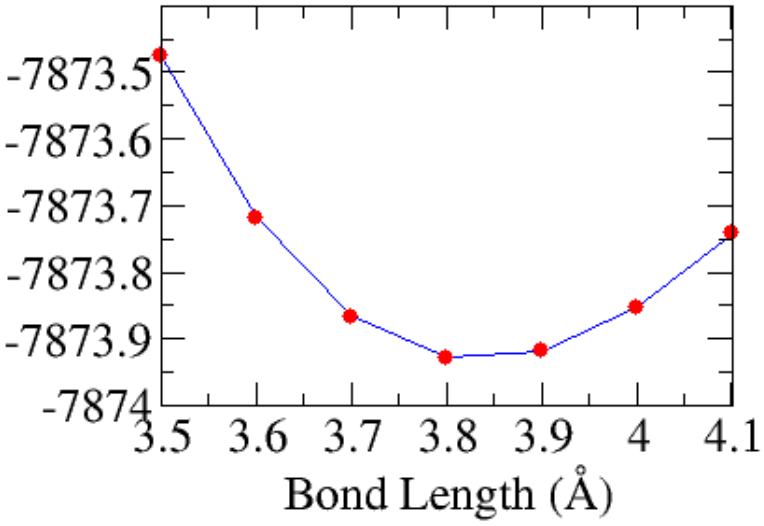


Figure 8: Bond length vs Total energy in fcc Si calculated with pw-lda.

Results

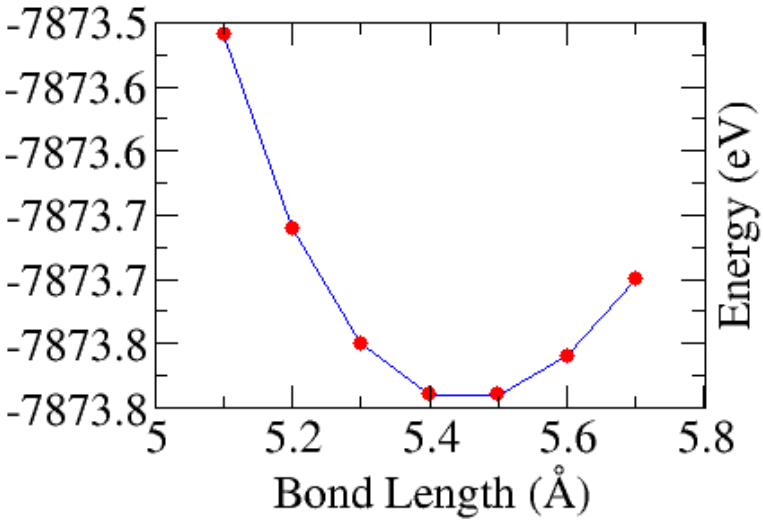


Figure 9: Bond length vs Total energy in diamond Si calculated with pw-lda.

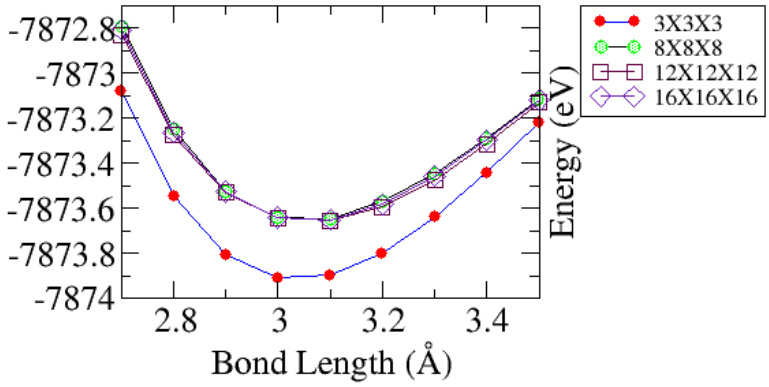


Figure 10: Bond length vs Total energy in bcc Si calculated with pw-lda for k-point convergence test.

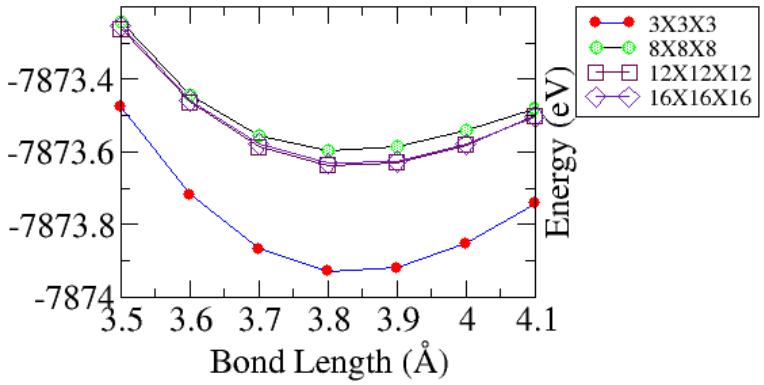


Figure 11: Bond length vs Total energy in fcc Si calculated with pw-lda for k-point convergence test.

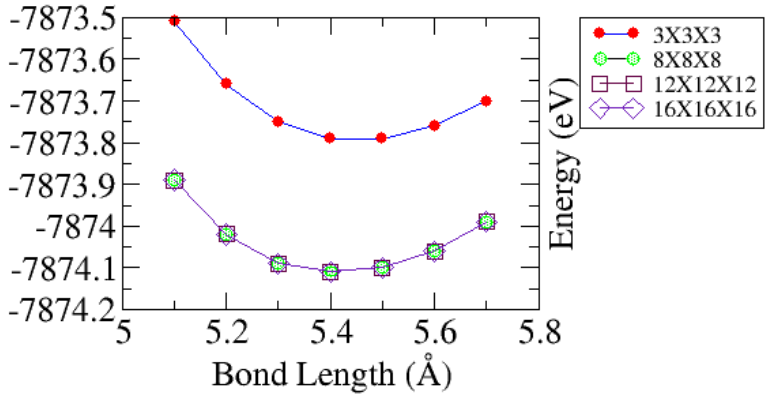


Figure 12: Bond length vs Total energy in diamond Si calculated with pw-lda for k-point convergence test.

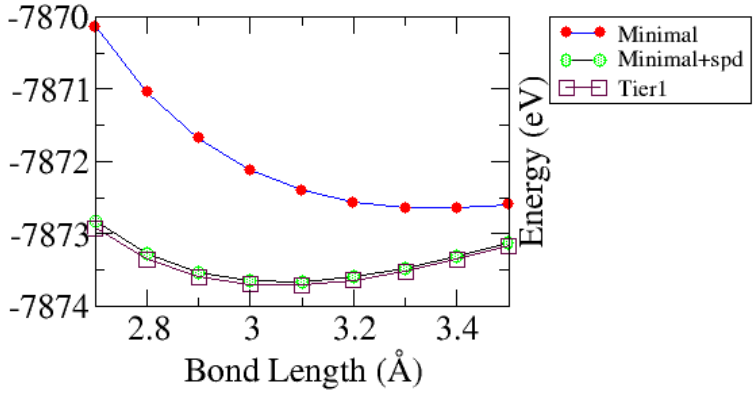


Figure 13: Bond length vs Total energy in bcc Si calculated with pw-lda for basis set convergence test with k-grids of 12×12×12..

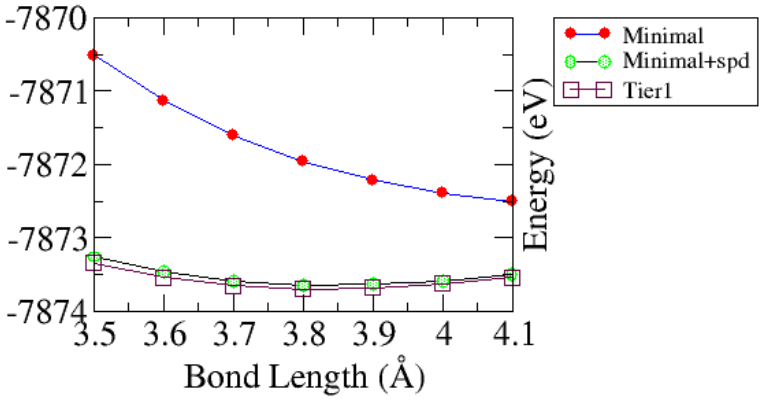


Figure 14: Bond length vs Total energy in fcc Si calculated with pw-lda for basis set convergence test with k-grids of 12×12×12..

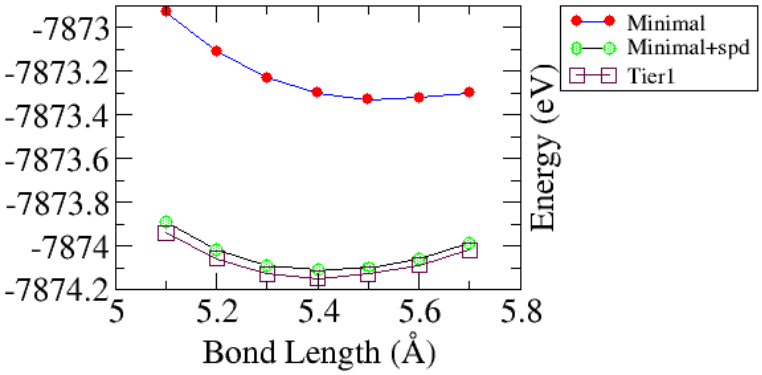


Figure 15: Bond length vs Total energy in diamond Si calculated with pw-lda for basis set convergence test with k-grids of 12×12×12.

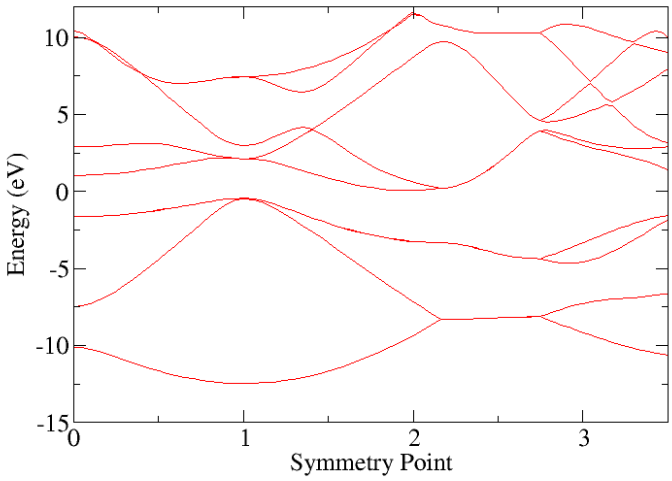


Figure 16: Band structure of diamond Si calculated with pw-lda with k-grids of 12×12×12.

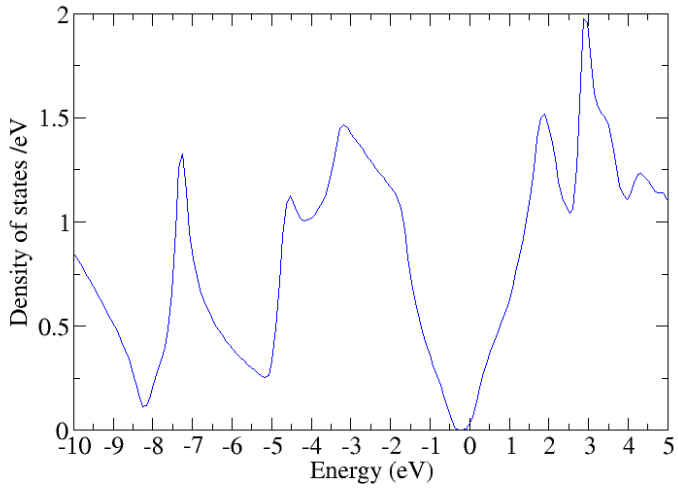


Figure 17: Density of states of diamond Si calculated with pw-lda with k-grids of 12×12×12.

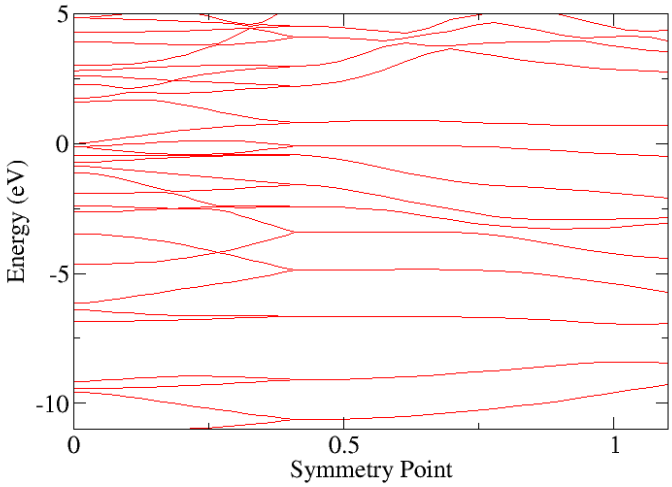


Figure 18: Band structure of 2×1 Si(001) surface (4 layers) calculated with pw-lda with k-grids of 12×24×1.

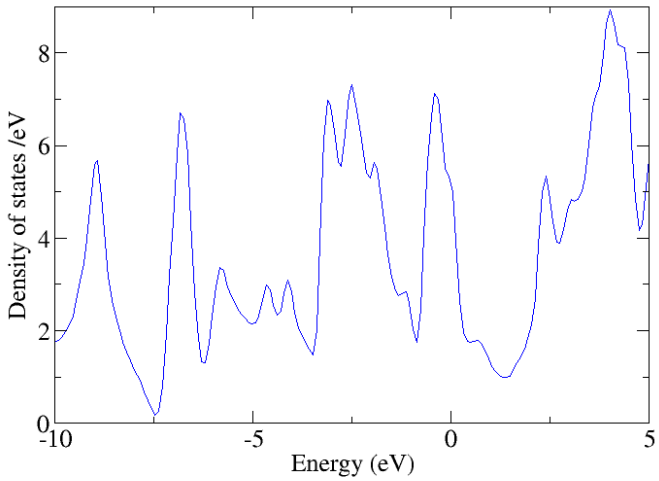


Figure 19: Density of states of 2×1 Si(001) surface (4 layers) calculated with pw-lda with k-grids of 12×24×1.

Appendix I: Structure information

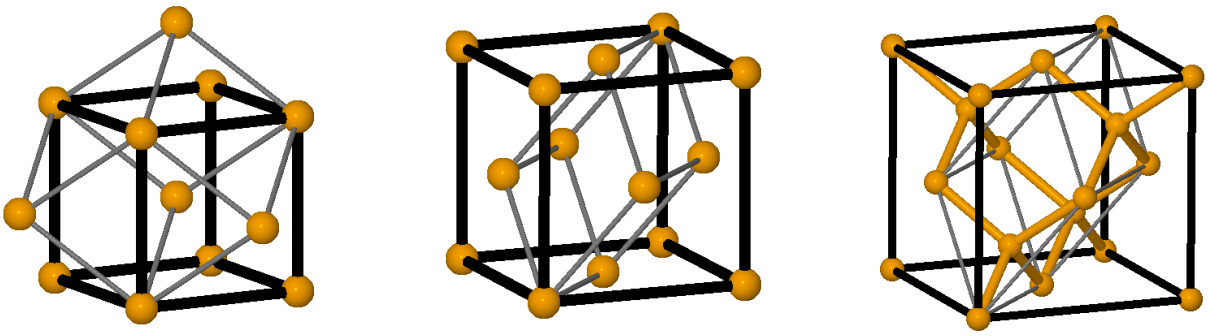


Figure 20: Cubic (black) and primitive (grey) unit cell for *bcc*, *fcc* and diamond Si (from left to right).

crystal structures	atomic coordinates	lattice vectors
fcc	$0 \quad 0 \quad 0$	$0 \quad a/2 \quad a/2$ $a/2 \quad 0 \quad a/2$ $a/2 \quad a/2 \quad 0$
diamond	$0 \quad 0 \quad 0$ $a/4 \quad a/4 \quad a/4$	$0 \quad a/2 \quad a/2$ $a/2 \quad 0 \quad a/2$ $a/2 \quad a/2 \quad 0$
bcc	$0 \quad 0 \quad 0$	$-a/2 \quad a/2 \quad a/2$ $a/2 \quad -a/2 \quad a/2$ $a/2 \quad a/2 \quad -a/2$
zincblende with atom species A and B	A $0 \quad 0 \quad 0$ B $a/4 \quad a/4 \quad a/4$	$0 \quad a/2 \quad a/2$ $a/2 \quad 0 \quad a/2$ $a/2 \quad a/2 \quad 0$

Table 1: Solids: Atomic coordinates and lattice vectors for different crystal structures.

diamond(001)					
atomic coordinates			lattice vectors		
0	0	0	$a/\sqrt{2}$	0	0
$a/2\sqrt{2}$	0	$a/4$	0	$a/\sqrt{2}$	0
$a/2\sqrt{2}$	$a/2\sqrt{2}$	$a/2$	0	0	L
0	$a/2\sqrt{2}$	$3a/4$			

Table 2: Surfaces: Atomic coordinates of an ideal diamond(001) surface. *Note:* a is the bulk lattice constant and L is the total slab thickness ($L = a + L_{\text{vac}}$ with the vacuum size L_{vac}).

Appendix II: High symmetry k-points

fcc	x_1	x_2	x_3
L	0.5	0.5	0.5
Γ	0	0	0
X	0	0.5	0.5
W	0.25	0.5	0.75
K	0.375	0.375	0.75

fcc(001)	x_1	x_2	x_3
$\bar{\Gamma}$	0.0	0.0	0.0
$\bar{J}$	0.5	0.0	0.0
$\bar{K}$	0.5	0.5	0.0

Table 3: High symmetry points for *fcc*/diamond bulk and (001) surface structures given in units of the three reciprocal lattice vectors ($\boldsymbol{k} = x_1\boldsymbol{b}_1 + x_2\boldsymbol{b}_2 + x_3\boldsymbol{b}_3$). In the case of *fcc*/diamond bulk, the reciprocal lattice vectors form a *bcc* structure and the corresponding lattice vectors can be found in Tab. 1.

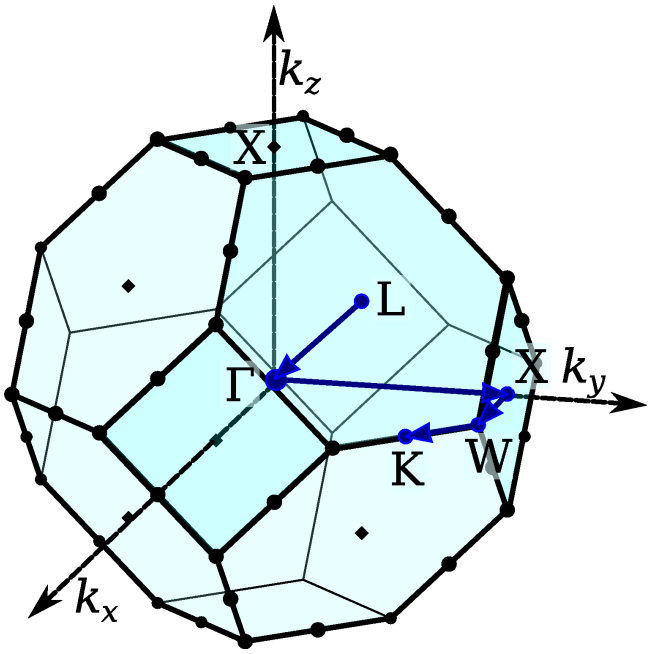


Figure 21: Brillouin zone and high symmetry points for *fcc*/diamond structure. The three coordinate axes ($\boldsymbol{k}_x$, $\boldsymbol{k}_y$, and $\boldsymbol{k}_z$) form a Cartesian coordinate system – not to be confused with the reciprocal lattice vectors of the *fcc*/diamond structure.

Appendix III: Python scripts

```python
#!/usr/bin/python
import os
import shutil
import numpy as np

#------------------------------------------------------------
#-------------Change script here-----------------------------
#------------------------------------------------------------

# Enter your calculated value for the minimal lattice constant
aMin = 5.1
# Enter your calculated value for the maximal lattice constant
aMax = 5.7
# Enter a value for the stepwidth
step = 0.1
# Number of calculations
n = int(np.rint((aMax - aMin)/step)) + 1

vector = np.zeros([3,3])
# Lattice vector in units of the lattice constant
vector[0] = [0.0,0.5,0.5]
vector[1] = [0.5,0.0,0.5]
vector[2] = [0.5,0.5,0.0]

#Number of basis atoms
atom_no = 2

atoms = np.zeros([atom_no,3])

#Basis atoms in fractional coordinates
atoms[0] = [0.0,0.0,0.0]
atoms[1] = [0.25,0.25,0.25]

#------------------------------------------------------------
#-------------End change script here-------------------------
#------------------------------------------------------------

for i in range(n):
   aLat = round((aMin + step * i),1)
   print("Processing lattice constant %10.6f AA." % aLat)
   # Create directory
   dirname = str(aLat)
   if not os.path.exists(dirname):
     os.makedirs(dirname)
   # Change to directory
   os.chdir(dirname)
   lattice_vector = vector * aLat
   # Write geometry.in
   filename = "geometry.in"
   f = open(filename,'w')
   # The lattice
   for lat in lattice_vector :
        f.write("lattice_vector {:10.6f} {:10.6f} {:10.6f}\n".format(lat[0],lat[1],lat[2]))
   # The atoms
   for a in atoms:
        f.write("atom_frac {:10.6f} {:10.6f} {:10.6f} Si\n".format(a[0],a[1],a[2]))
   # Close file
   f.close()

   # Copy the control file
   shutil.copyfile("../control.in","control.in")

   # Run FHI-aims on 4 processes
   #os.system("mpirun -n 4 aims.x > aims.out")

   # Change back to former directory
   os.chdir("..")
```

Figure 22: Example python script for running calculations for several lattice constants used in Part I (`skel/problem_1/02_3x3x2/diamond/run_diamond.py`).

```python
#!/usr/bin/python
import os
import numpy as np

# Minimal lattice constant in A (from run script)
aMin = 5.1
# Maximal lattice constant in A (from run script)
aMax = 5.7
# Sampling density in A (from run script)
step = 0.1
#Number of basis atoms
atom_no = 2

# Number of calculations
n = int(np.rint((aMax - aMin)/step)) + 1

data = open("energies.dat",'w')
data.write("#%22s   %20s\n" % ("lattice constant (AA)","energy (eV/atom)"))

for i in range(n):
   aLat = round((aMin + step * i),1)
   print("Postprocessing lattice constant %10.6f AA." % aLat)
   dirname = str(aLat)
   filename = dirname + "/" + "aims.out"
   # Check if calculation was running
   if (not os.path.isfile(filename)):
      print("%s was not processed." % filename)
   else:
      # Check for convergence
      f = open(filename,'r')
      converged = False
      for line in f:
         if "Have a nice day" in line:
            converged = True
         if "Self-consistency cycle converged." in line:
            converged = True
      f.close()

      if (not converged):
         print("%s is not converged." % filename)
      else:
         # Grep for total energy
         f = open(filename,'r')
         for line in f:
            if "Total energy corr" in line:
               linesplit = line.split()
               energy = float(linesplit[5])/float(atom_no)
               data.write(" %16.6f   %23.6f\n" % (aLat,energy))
         f.close()
data.close()
```

Figure 23: Example python script to retrieve total energies from FHI-aims output files used in Part I (`skel/problem_1/02_3x3x2/diamond/postprocess.py`).

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

import sys
from math import sqrt

def output_lattice_vector(x, y, z):
    print("lattice_vector  %.16f  %.16f  %.16f" % (x, y, z))
def output_atom(x, y, z, name):
    print("atom %.16f  %.16f  %.16f  %s" % (x, y, z, name))
def output_constrained_atom(x, y, z, name):
    print("atom %.16f  %.16f  %.16f  %s" % (x, y, z, name))
    print("  constrain_relaxation .true.")

A = 5.416            # Lattice constant
L_vac = 30.          # Vacuum
A_1x1 = A/sqrt(2.)   # 1x1 surface periodicity
n_layer = 4          # Number of layers in z-direction
Z = A/4.             # Layer separation in z-direction
C = 0.5 * A_1x1      # Row separation in x- and y-direction

# H-saturation is put at this fraction of where the next
# Si atom would have been.
frac_H = 0.63

# (2x1) reconstructed lattice:
output_lattice_vector(2*A_1x1, 0.,    0.)
output_lattice_vector(0.,      A_1x1, 0.)
output_lattice_vector(0.,      0.,    n_layer*Z+L_vac)
print

# Hydrogen saturation
# The next Si would have been at (+/-C, 0., -Z).
output_atom(   -frac_H*C, 0., -frac_H*Z, "H")
output_atom(   +frac_H*C, 0., -frac_H*Z, "H")
# The next Si would have been at (2*C+/-C, 0., -Z).
output_atom(2*C-frac_H*C, 0., -frac_H*Z, "H")
output_atom(2*C+frac_H*C, 0., -frac_H*Z, "H")
# Bottom Si layer
output_atom(0*C, 0., 0*Z, "Si")
output_atom(2*C, 0., 0*Z, "Si")
# Other Si layers
output_atom(0*C, C,  1*Z, "Si")
output_atom(2*C, C,  1*Z, "Si")
output_atom(1*C, C,  2*Z, "Si")
output_atom(3*C, C,  2*Z, "Si")
output_atom(1*C, 0., 3*Z, "Si")
output_atom(3*C, 0., 3*Z, "Si")
```

Figure 24: The python script used in Part III (`skel/problem_6/01_ideal_2x1_4_layers/write-geom.py`). The script creates a geometry output (which can be redirected to a file) for the ideal hydrogen saturated 2×1 Si(001) surface with 4 layers.

Acknowledgments

We would like to thank all the aims developers and tutors of aims workshops.

References

[1] M. T. Yin and M. L. Cohen, *Microscopic Theory of the Phase Transformation and Lattice Dynamics of Si*, Phys. Rev. Lett. **45**, 1004 (1980).

[2] C. Kittel, *Introduction to Solid State Physics* (John Wiley & Sons Inc, 1986), 6th edition.

[3] F. D. Murnaghan, *The compressibility of media under extreme pressures*, Proc. Natl. Acad. Sci. **30**, 244 (1944).

[4] F. Birch, *Finite Elastic Strain of Cubic Crystals*, Phys. Rev. **71**, 809 (1947).

[5] W. Setyawan and S. Curtarolo, *High-throughput electronic band structure calculations: Challenges and tools*, Comput. Materials Science **49**, 299 (2010).

[6] R. A. Wolkow, *Direct observation of an increase in buckled dimers on Si(001) at low temperature*, Phys. Rev. Lett. **68**, 2636 (1992).

[7] W. P. Huhn and V. Blum, *One-hundred-three compound band-structure benchmark of post-self-consistent spin-orbit coupling treatments in density functional theory*, Phys. Rev. Materials **1**, 033803 (2017).