PHONON CALCULATION

In this tutorial, you will learn how to perform a phonon calculation with `FHI-vibes` for fcc Silicon.

Set up phonon calculation

Setting up a `phonopy` calculation is similar to setting up a `relaxation` (or any other workflow supported by `FHI-vibes`): we will need a geometry file and the phonopy calculation settings' file. To this end, first change into the empty directory `02_phonons/input/sc_0002` via

```
cd 02_phonons/input/sc_0002
```

There, we will perform the phonon calculation on the geometry-optimized structure from the previous step, which we need to copy over to the current working dirctory and rename:

```
cp path/to/previous/workdir/relaxation/geometry.in.next_step geometry.in
```

Similarly, we copy over the settings for `FHI-aims` and use them as basis for setting up the phonon workflow. To this end, execute

```
cp path/to/previous/workdir/calculator.in phonopy.in
```

N.B. For your convenience, a copy of `calculator.in` is already present in the folder as well. But make sure that it matches your earlier settings. This is important, since we need to ensure that our `FHI-aims` calculator settings do not change. Now, we add settings for a `phonopy` calculation by running

```
vibes template phonopy >> phonopy.in
```

The `phonopy.in` now reads as:

```
[files]
geometry:                       geometry.in

[calculator]
name:                           aims
socketio:                       True

[calculator.parameters]
xc:                             pw-lda
compute_forces:                 true

[calculator.kpoints]
density:                        2

[calculator.basissets]
default:                        light

[phonopy]
supercell_matrix:               [1, 1, 1]
displacement:                   0.01
is_diagonal:                    False
is_plusminus:                   auto
symprec:                        1e-05
q_mesh:                         [45, 45, 45]
workdir:                        phonopy
```

The newly added `[phonopy]` section contains information about how the supercells with displacements should be set up to compute the force constants via the finite-differences method. Behind the scenes, this steps relies on the Phonopy package. An explanation for the full list of keywords is found in the `FHI-vibes` documentation. The two most important keywords are explained in the following.

Supercell Matrix (`supercell_matrix`)

The supercell matrix M_S given as `supercell_matrix` will be used to generate the lattice of the supercell from the lattice of the primitive unitcell by matrix multiplication:

$$\begin{pmatrix} \mathbf{a}_S^\mathrm{T} \\ \mathbf{b}_S^\mathrm{T} \\ \mathbf{c}_S^\mathrm{T} \end{pmatrix} = M_s \cdot \begin{pmatrix} \mathbf{a}_u^\mathrm{T} \\ \mathbf{b}_u^\mathrm{T} \\ \mathbf{c}_u^\mathrm{T} \end{pmatrix}$$

Here, $\mathbf{a}_u^t$, $\mathbf{b}_u^t$, $\mathbf{c}_u^t$ are the transposed lattice vectors (row-vectors) of the (primitive) unit cell and $\mathbf{a}_S$, $\mathbf{b}_S$, $\mathbf{c}_S$ label the lattice vectors of the supercell respectively.
`supercell_matrix` can be given in any shape that lets itself transform trivially to a 3×3-matrix. For example, `[1, 1, 1]` gets transformed to the 3×3 unit matrix.

Displacement (`displacement`)

The `displacement` tag will set the amplitude of the finite displacement in Å. In principle, this is a numerical parameter that needs to be optimized. A smaller `displacement` results in a better approximation to the true second derivative of the potential. However, too small of a displacement generates forces that are too small and can be severely affected by other sources of computational noise, e.g., finite grids, etc. For production purposes, the default value of $d = 0.01\text{Å}$ usually works quite well and there is no need to increase the displacement further with a properly set up force calculator.

Run the calculation

Let's stick to the default settings in `phonopy.in` for the moment and run the calculation with

```
vibes run phonopy | tee log.phonopy
```

The calculation should take only a few seconds (depending on you computer).

Postprocessing

The `vibes run` command takes care that all ab initio calculations are performed, but some additional postprocessing is needed to obtain the phonon-related quantities. The postprocessing itself can be performed interactivley with

```
vibes output phonopy phonopy/trajectory.son --full
```

where the output will look like:

```
[phonopy.postprocess] Start phonopy postprocess:
[trajectory]   Parse `phonopy/trajectory.son`
[son] open file:  phonopy/trajectory.son
[trajectory]   .. time elapsed: 0.004s
[trajectory]   .. create atoms
[trajectory]   .. time elapsed: 0.001s
[trajectory]   .. set raw hash
[trajectory]   .. time elapsed: 0.000s
[trajectory]   .. done in 0.005s
[phonopy.postprocess] .. done in 1.113s
q_mesh not given, use values stored in phonopy/trajectory.son: [45, 45, 
[phonopy.postprocess]
Extract phonopy results:
[phonopy.postprocess] .. q_mesh:   [45, 45, 45]
[phonopy.postprocess] .. write force constants
[phonopy.postprocess] Extract basic results:
[phonopy.postprocess] .. write primitive cell
[phonopy.postprocess] .. write supercell
[phonopy.postprocess] .. write force constants to FORCE_CONSTANTS
[phonopy.postprocess] Extract bandstructure
[phonopy.postprocess] .. write yaml
[phonopy.postprocess] Run mesh
[phonopy.postprocess] Extract thermal properties
[phonopy.postprocess] .. write yaml
[phonopy.postprocess] Extract DOS:
[phonopy.postprocess] .. write
[phonopy.postprocess] .. animation/animation_G.ascii written
[phonopy.postprocess] .. animation/animation_K.ascii written
[phonopy.postprocess] .. animation/animation_L.ascii written
[phonopy.postprocess] .. animation/animation_U.ascii written
[phonopy.postprocess] .. animation/animation_W.ascii written
[phonopy.postprocess] .. animation/animation_X.ascii written
[phonopy.postprocess] .. all files written to phonopy/output in 376.936s
* Message from file vibes/phonopy/postprocess.py, line 131, function che
--> Negative frequencies found at G = [0 0 0]:

# Mode   Frequency
    1 -9.47776e-08 THz
[phonopy.postprocess]
Plot phonopy results:
[phonopy.postprocess] Plot thermal properties
[phonopy.postprocess] Plot bandstructure
[phonopy.postprocess] Plot DOS:
[phonopy.postprocess] .. all files written to phonopy/output in 785.225s
```

This will:

- Compute the phonon bandstructure along high symmetry paths in the Brillouin zone and

save it in `phonopy/output/bandstructure.pdf`.

- Compute the density of states using a 45×45×45 $\mathbf{q}$ point grid and the Tetrahedron method. The density of states will be plotted alongside the bandstructure to a file `output/bandstructure_dos.pdf`, and written to a data file `total_dos.dat`. The q-grid can be adjusted by specifying it with an additional flag `--q_mesh`.
- Compute the harmonic free energy F^{ha} and the harmonic heat capacity at constant volume, C_V, i.e., the thermal properties accessible in the harmonic approximation using the DOS and its q-point settings. An overview plot is saved to `output/thermal_properties.pdf` and the detailed output is written to `output/thermal_properties.yaml`.
- Write a `phonopy.yaml` for loading the Phonopy object directly within `python`.
- You can also visualize the actual phonons using the following commands: `cd phonopy/output phonopy --band auto --eigenvectors` In this directory, this will create a `band.yaml` file including the mode eigenvectors on a default Brillouin zone path. Then go to Henrique Miranda's amazing phonon website and switch to the animation page. Load your band.yaml file via "Custom File: Browse".

Congratulations! You have just performed a full (but not yet converged!) ab initio phonon bandstructure calculation.

Note that the CLI also allows to only run a subset of the postprocessing, e.g.,

```
vibes output phonopy phonopy/trajectory.son -v -bs
```

only outputs the bandstructure.

Choosing a supercell size

There is a CLI utility in `FHI-vibes` that can help you to find supercells of different sizes:

```
vibes utils make-supercell
```

For example

```
vibes utils make-supercell geometry.in -n 8
```

will find the conventional, cubic cell of silicon with 8 atoms:

```
Find supercell for
[vibes]        Geometry info
  input geometry:    Si
  Symmetry prec.:    1e-05
  Number of atoms:   2
  Species:           Si (2)
  Periodicity:       [ True  True  True]

  Spacegroup:          Fd-3m (227)
  Wyckoff positions:   2*b
  Equivalent atoms:    2*0


Settings:
  Target number of atoms: 8

Supercell matrix:
 python:  [-1,  1,  1,  1, -1,  1,  1,  1, -1]
 cmdline: -1 1 1 1 -1 1 1 1 -1
 2d:
[[-1, 1, 1],
 [1, -1, 1],
 [1, 1, -1]]

Superlattice:
[[5.44 0.   0.  ]
 [0.   5.44 0.  ]
 [0.   0.   5.44]]

Number of atoms:  8
  Cubicness:               1.000 (1.000)
  Largest Cutoff:          2.720 AA
  Number of displacements: 1 (1)

Supercell written to geometry.in.supercell_8
[trajectory]   .. time elapsed: 1.337s
```

It will tell you the supercell matrix that you can use in `phonopy.in` (`python: [-1, 1, 1, 1, -1, 1, 1, 1, -1]`), the generated superlattice, a "cubicness" score based on the filling ratio of the largest sphere fitting into the cell, the largest cutoff in which any neighbor is not a periodic image of a closer neighbor to estimate boundary effects, and the number of supercells with displacements that `phonopy` will create. It will also write the structure to `geometry.in.supercell_8` which you can inspect.

To test a different supercell size, copy your old calculation to a new folder and remove the old data via:

`cd .. && cp -r sc_0002 sc_0008 && cd sc_0008 && rm -rf log.phonopy phonopy`
Now, one just needs to replace the `supercell` keyword in the `[phonopy]` section with the required value:

```
...
[phonopy]
supercell_matrix: [-1,  1,  1,  1, -1,  1,  1,  1, -1]
...
```

and run `vibes run phonopy | tee log.phonopy` and `vibes output phonopy phonopy/trajectory.son --full` again.

Now, compare the band structures that you can find in the respective `phonopy/output` folders. Do you notice something? Can you explain the differences?

OPTIONAL: If interested in phonons, you might want trying different supercells settings, e.g.,

```
...
[phonopy]
supercell_matrix: [2,  2,  2]
...
```

or

```
...
[phonopy]
supercell_matrix: [-2,  2,  2,  2, -2,  2,  2,  2, -2]
...
```

Remember to use a new working directory to not mess up your previous results! Alternatively, you can look at the reference calculations provided in the solutions folder.

Practical guideline

In practice, the convergence with supercell size needs always to be checked carefully, since it depends on the range of the interactions present in your system, e.g., long ranged unscreened van-der-Waals interactions require larger supercells than short-ranged covalent ones as here in Si. Along the same lines, the acceptable supercell size depends also on the properties you are interested in. Free energies and specific heats converge faster than individual phonon frequencies. Using a supercell that is as cubic as possible in shape, playing around with `vibes utils make-supercell`, and a little bit of experience will do the job. For the example at hand, it might, for instance, be instructive to check the convergence of different properties for larger supercell sizes. You find calculations for supercell sizes up 1,728 atoms in our tutorial repository under solutions. When do you

consider the calculations as converged?