Phonons in Siesta-tutorial

Diego Juarez

October 31th, 2025

Contents

1 Silicon

Notes to document my notes running Phonons-tutorial from `https://github.com/siesta-project/tutorials`. First I clone the directory with the tutorials

Listing 1: Clone SIESTA

```
# Clone the official SIESTA tutorials repository
git clone https://github.com/siesta-project/tutorials.git

# Enter the Phonons tutorial directory
cd tutorials/Phonons

# (Optional) see if your documents are present
ls -1
```

Now I wrote a python script to run the commands as I installed Siesta Sofware in my own environment. I let the steps how to create, activate your environment and install it (conda environment)

Listing 2: Create and activate a Conda environment with SIESTA and phonon tools

```
# (1) Create env (uses conda-forge)
conda create -n siesta-env -c conda-forge \
  siesta openmpi

# (1.5) Optional if are not installed
conda install libxc libnetcdf netcdf-fortran \
  python=3.11 numpy scipy matplotlib ase spglib phonopy \
  pip -y

# (2) Activate env
conda activate siesta-env

# (3) Verify software
siesta --version
```

Following is my python script which run the commands to create the output files to generate the phonon dispersion:

```
import os
import subprocess

num_processors = 6
input_path = "/global/cfs/projectdirs/m3845/Diego_projects/AIMD-siesta/tutorials/Phonons/Silicon/"
siesta_exec = "/global/homes/d/dajuarez/.conda/envs/siesta_env/bin/siesta"
os.environ["PATH"] = os.path.dirname(siesta_exec) + ":" + os.environ.get("PATH", "")
os.chdir(input_path)

cmd2 = 'fcbuild < si54.fdf > fcbuild.log'
subprocess.run(cmd2, shell=True, check=True)

cmd3 = f'mpirun -np {num_processors} {siesta_exec} < si54-siesta.fdf | tee output.txt'
subprocess.run(cmd3, shell=True, check=True)

cmd4 = 'vibra < si54.fdf > vibra.out'
subprocess.run(cmd4 , shell=True, check=True, cwd=input_path)

cmd5 = 'gnubands < si54.bands > si54_bands.gpl'
subprocess.run(cmd5, shell=True, check=True)
```

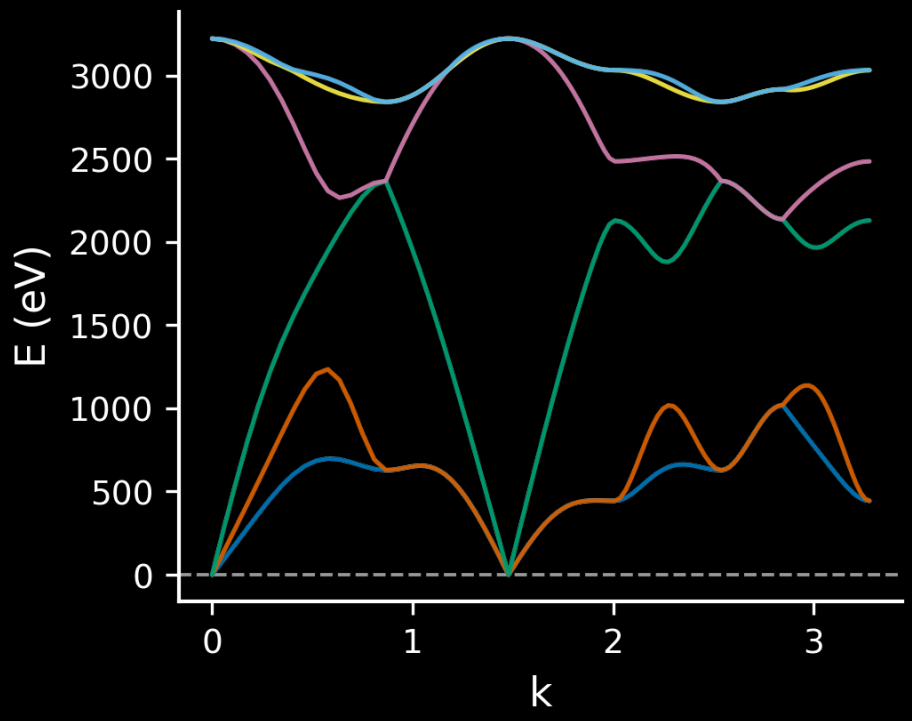
And to plot the output file

```
import re, sys, math
from pathlib import Path
import matplotlib.pyplot as plt

INPUT_PATH = Path("si54_bands.gpl")
OUTPUT_PNG = Path("bands_plot.png")
OUTPUT_SVG = Path("bands_plot.svg")
```

```
line_w = 1.2
alpha = 0.95
OKABE_ITO = ["#0072B2","#D55E00","#009E73","#CC79A7",
            "#F0E442","#56B4E9","#E69F00","#000000"]
# --------------------------

def parse_blocks(text):
    segments, current = [], []
    ef = 0.0
    m = re.search(r"E_F\s*=\s*([+-]?\d+(?:\.\d*)?(?:[eE][+-]?\d+)?)", text)
    if m: ef = float(m.group(1))

    for raw in text.splitlines():
        s = raw.strip()
        if not s or s.startswith('#') or set(s) == {'-'}:
            if current:
                segments.append(current); current = []
            continue
        vals = []
        for p in s.split():
            try: vals.append(float(p))
            except ValueError: pass
        if len(vals) >= 2 and math.isfinite(vals[0]) and math.isfinite(vals[1]):
            k, E = vals[0], vals[1]
            current.append((k, E))
        else:
            if current: segments.append(current); current = []
    if current: segments.append(current)
    return segments, ef

def load_text(path: Path) -> str:
    if path.exists(): return path.read_text(encoding="utf-8", errors="ignore")
    if not sys.stdin.isatty(): return sys.stdin.read()
    raise FileNotFoundError(f"Missing '{path}'. Provide file or pipe stdin.")

def main():
    text = load_text(INPUT_PATH)
    segments, ef = parse_blocks(text)
    if not segments: raise RuntimeError("No numeric data segments found.")

    plt.rcParams.update({
        "figure.dpi": 300,
        "savefig.dpi": 300,
        "font.size": 10,
        "axes.labelsize": 11,
        "axes.titlesize": 11,
        "legend.fontsize": 9,
        "xtick.labelsize": 9,
        "ytick.labelsize": 9,
        "axes.spines.top": False,
        "axes.spines.right": False,
        "axes.linewidth": 1.0,
        "mathtext.default": "regular",
    })

    fig, ax = plt.subplots(figsize=(3.5, 2.8))

    ax.set_prop_cycle(color=OKABE_ITO)

    for i, seg in enumerate(segments):
        ks = [p[0] for p in seg]
        Es = [p[1] for p in seg] # eV; convert if needed before plotting
        ax.plot(ks, Es, lw=line_w, alpha=alpha, solid_joinstyle="round", solid_capstyle="round")

    ax.axhline(0.0, ls="--", lw=0.9, color="#7f7f7f", alpha=0.7, zorder=0)
```

```
    ax.set_xlabel(r"$k$ (path units)")
    ax.set_ylabel(r"$E$ (eV)")
    ax.grid(True, lw=0.3, alpha=0.25)
    ax.tick_params(direction="out", length=3.5, width=0.8)

    fig.tight_layout(pad=0.8)
    fig.savefig(OUTPUT_PNG, bbox_inches="tight")
    fig.savefig(OUTPUT_SVG, bbox_inches="tight")
    print(f"Saved: {OUTPUT_PNG.resolve()}\nSaved: {OUTPUT_SVG.resolve()}")

if __name__ == "__main__":
    main()
```

With the following plot:

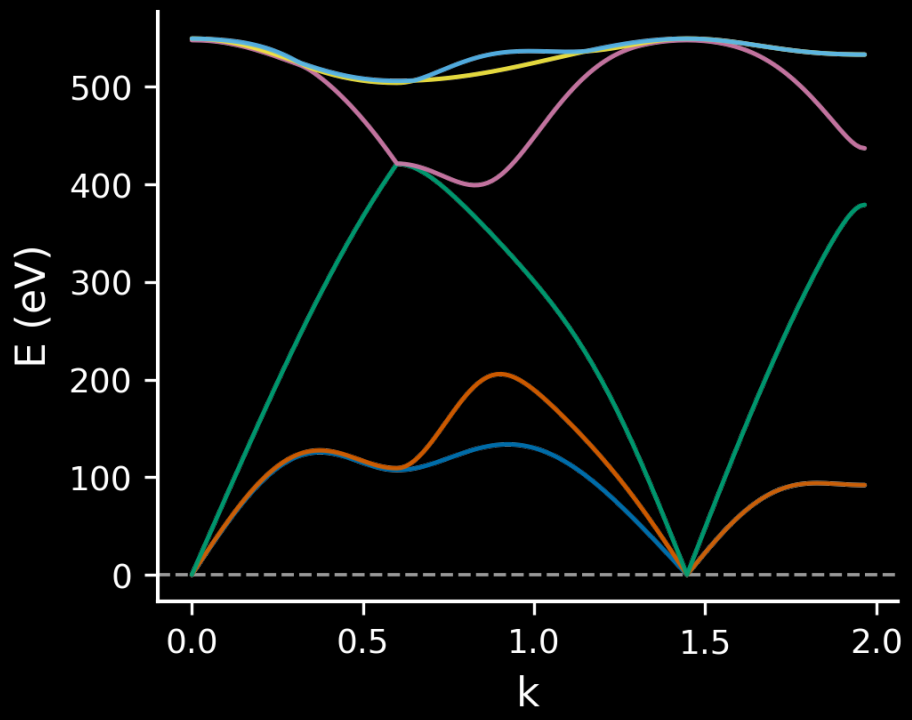


2 Silicium

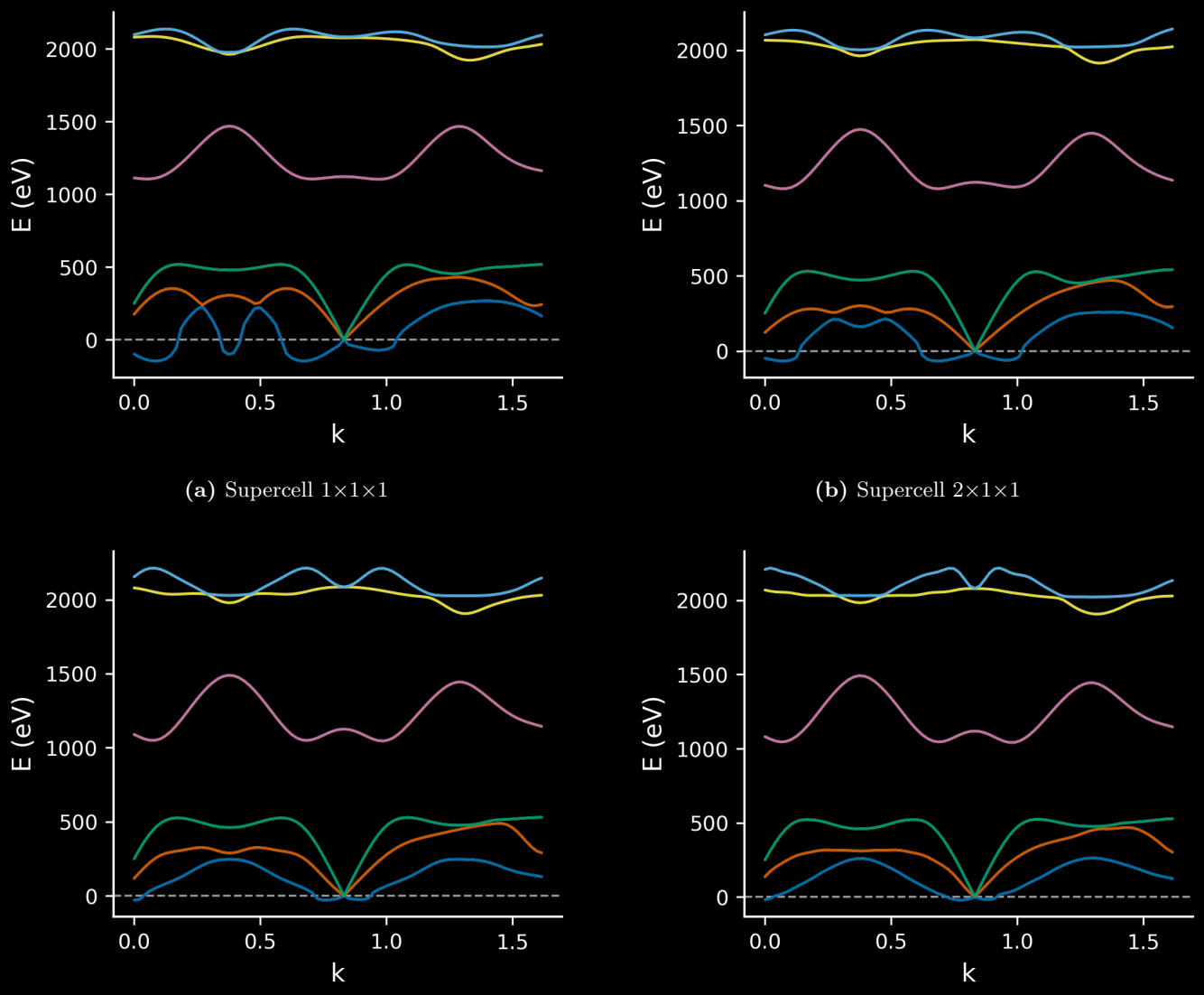
Now for the example labeled as *Si.new* with the following commands executed and python script as before we got

```
   import os
import subprocess
num_processors = 6
input_path = '/global/cfs/projectdirs/m3845/Diego_projects/AIMD-siesta/tutorials/Phonons/Si-new/'
siesta_exec = "/global/homes/d/dajuarez/.conda/envs/siesta_env/bin/siesta"
os.environ["PATH"] = os.path.dirname(siesta_exec) + ":" + os.environ.get("PATH", "")
os.chdir(input_path)
relax_file = 'Si.relax-structure.fdf'
build_file = 'Si.fcbuild.fdf'
int_atom_fc_file ='Si.ifc.fdf'
cmd1 = f"{siesta_exec} < {relax_file} > relaxed.log"
subprocess.run(cmd1, shell=True, check=True)
cmd2 = f'fcbuild < {build_file} | tee build.log'
subprocess.run(cmd2, shell=True, check=True)
cmd3 = f"{siesta_exec} < {int_atom_fc_file} > Si.ifc.111.log"
```

```
subprocess.run(cmd3, shell=True, check=True)
cmd4 = 'vibra < Si.fcbuild.fdf > vibra.out'
subprocess.run(cmd4 , shell=True, check=True, cwd=input_path)
cmd5 = 'gnubands < Si.bands > Si.phonon-bands.111.dat'
subprocess.run(cmd5, shell=True, check=True)
```



3 Gallium Arsenide - 2D



(a) Supercell 1×1×1

(b) Supercell 2×1×1

(c) Supercell 3×3×1

(d) Supercell 6×6×1

Figure 3: Phonon dispersion of two-dimensional Gallium Arsenide (GaAs) for different supercell sizes.

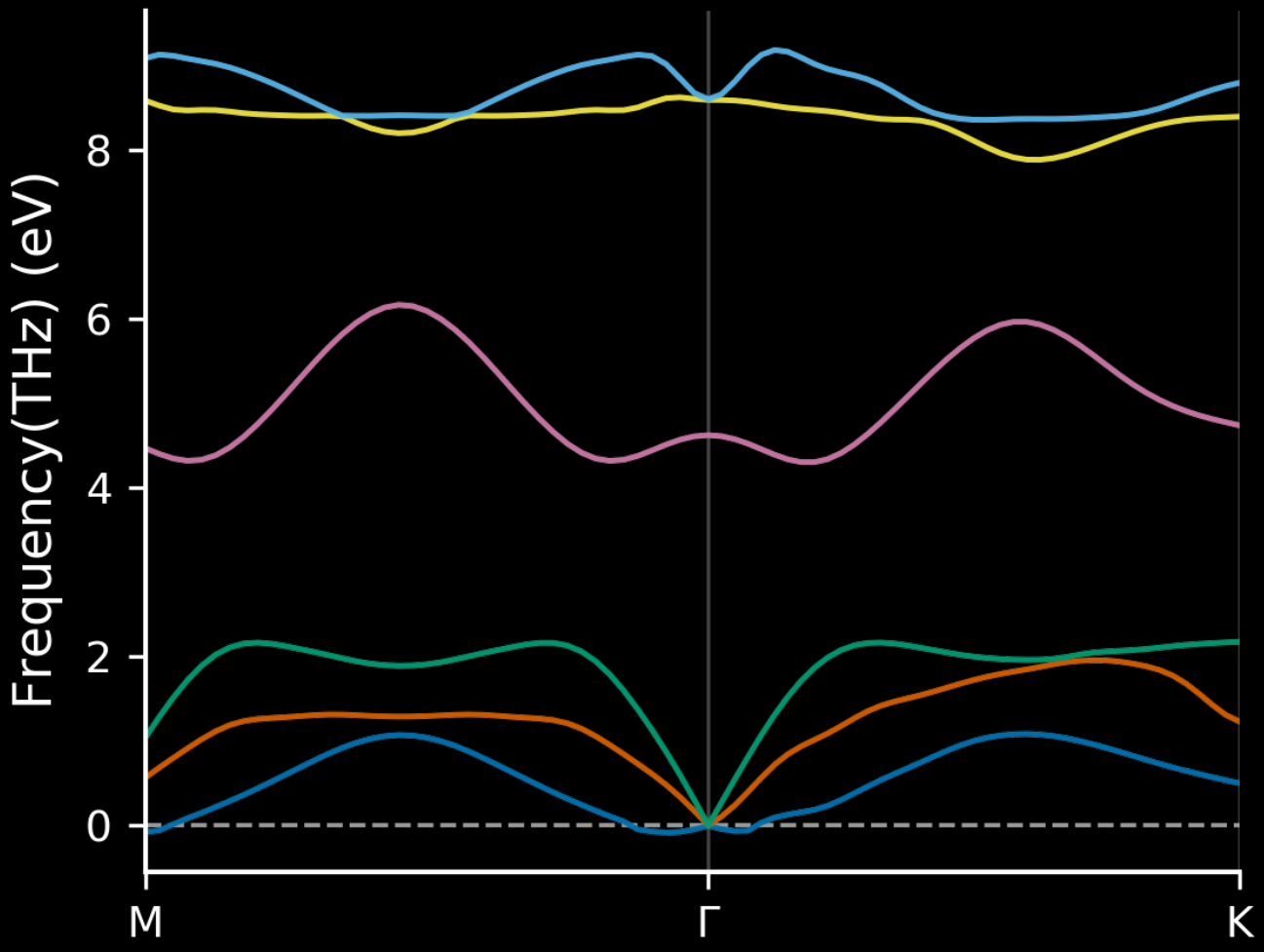


Figure 4: Supercell 8×5×1