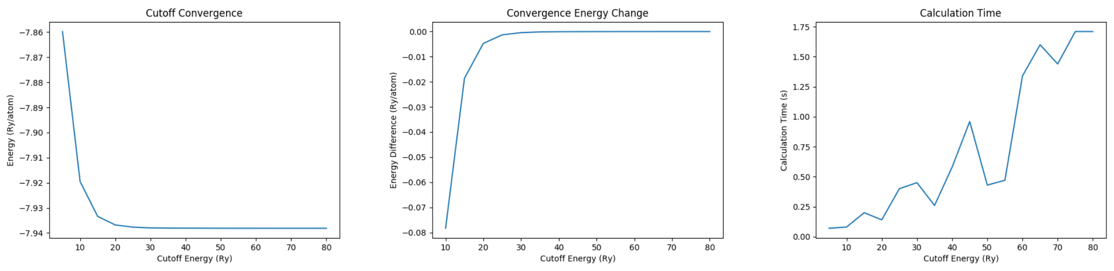
Computational Design of Materials Lab 2

Andrew Sullivan

March 17, 2019

1 Problem 1: Cutoff Convergence of Absolute Energies



(a) Energy. (b) Convergence of Energy. (c) Calculation Time.

Figure 1: Absolute Energy vs. Kinetic Energy Cutoff.

1.1 A

In this section, we examine the convergence behavior of the Quantum Espresso DFT code for a germanium lattice with diamond cubic structure with respect to the kinetic energy cutoff, which sets the maximum kinetic energy (and by extension, the largest multiple of the reciprocal lattice vector) included in the approximation. We use a fixed k-point grid of 4 x 4 x 4 by setting the nk variable equal to 4, a fixed lattice parameter of 5.0 Å, and vary the energy between 5 and 80 Ry in increments of 5 Ry. This leads to the convergence behavior in Figure 1. The energy converges to a value of -7.94 Ry/atom, and the second plot shows that the incremental change in energy decreases below 5 meV/atom (or 0.0004 Ry/atom) at a value of 35 Ry. Note than we define convergence as the difference between the final energy value and the energy value at each energy cutoff point.

1.2 B

The third plot contains the calculation time as a function of the cutoff energy, showing a clear positive relationship between the two. This is expected, as higher kinetic energy cutoffs means more values are included for each k-point, thereby increasing the total computational cost.

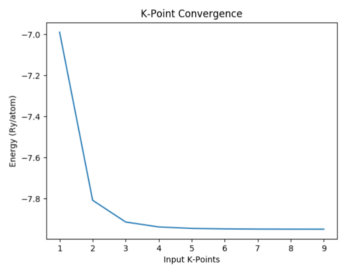
1.3 C

One advantage of using the primitive cell over the unit cell is obvious- the primitive cell contains less atoms (in this case, 2 vs. 8 for diamond cubic structure) and a smaller overall volume. In many cases, however, the symmetries present in the unit cell are not necessarily reflected in the primitive cell. Therefore, while a primitive cell may allow for fewer calculations due to a smaller number of atoms, a unit cell may alternatively facilitate fewer calculations through symmetries that may not otherwise be apparent. Symmetries can in turn reduce the number of k-points required to effectively sample the Brillouin zone.

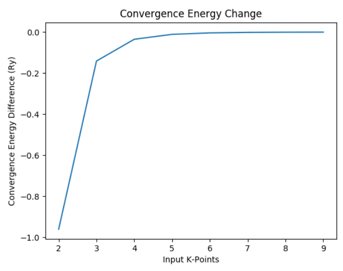
2 Problem 2: K-Point Convergence of Absolute Energies

2.1 A

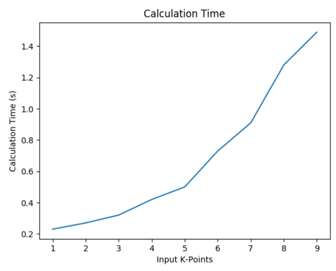
As before, we use a diamond cubic Ge lattice with a lattice parameter of 5.0 Å, and fix our kinetic energy cutoff to a value of 30 Ry, which should be nearly sufficient in light of the trend seen in the previous problem. The resultant energy is recorded for each input number of k-points, specified as the number in each direction (so an input value of 2 corresponds to a 2 x 2 x 2 grid), along with the computation time and the number of unique k-points, as some are removed due to the symmetries of the crystal. Figures 2 and 3 display these results as a function of the input grid size and the number of unique k-points, respectively.



(a) Energy.

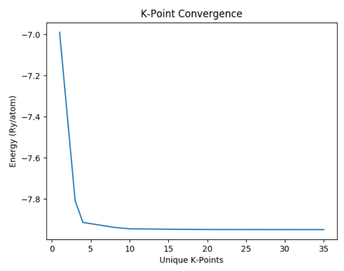


(b) Convergence of Energy.

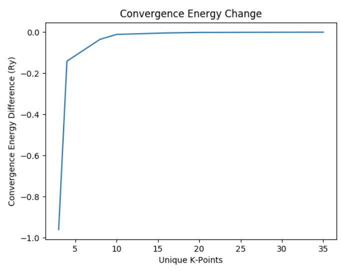


(c) Calculation Time.

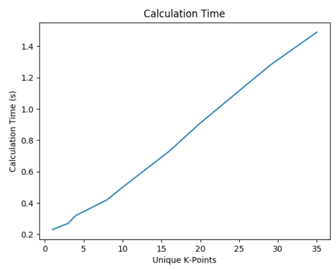
Figure 2: Absolute Energy vs. Input K-Point Grid Size.



(a) Energy.



(b) Convergence of Energy.



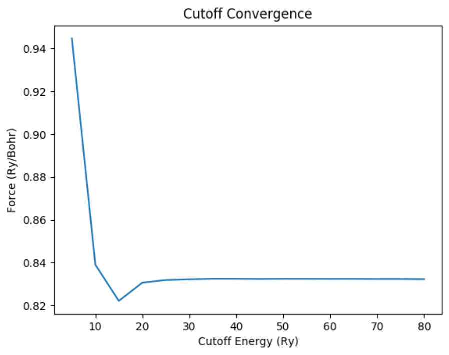
(c) Calculation Time.

Figure 3: Absolute Energy vs. Unique Number of K-Points.

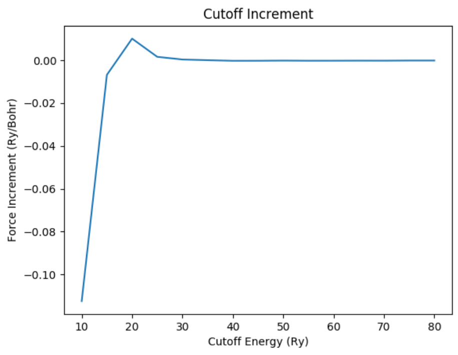
2.2 B

A trend is evident for both the calculated energies and the computation times. For the former, the energy converges to a constant value of -7.95 Ry/atom around 35 unique k-points, or an input grid of 9 x 9 x 9. Beyond this value, increasing the number of k-points does not appear to significantly change the approximation, as all of the major energy terms (e.g. Hartree, exchange-correlation, and kinetic energy) have been accounted for, though the computation time continues to increase since more k-points are still being tested, even though the result is not much better.

3 Problem 3: Cutoff Convergence of Forces



(a) Force.

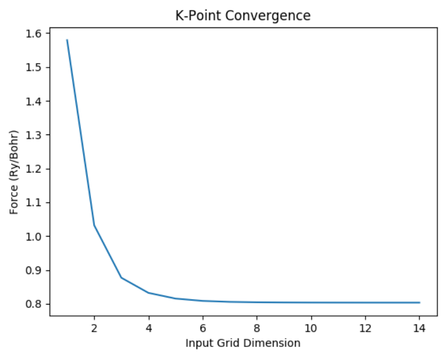


(b) Convergence of Force.

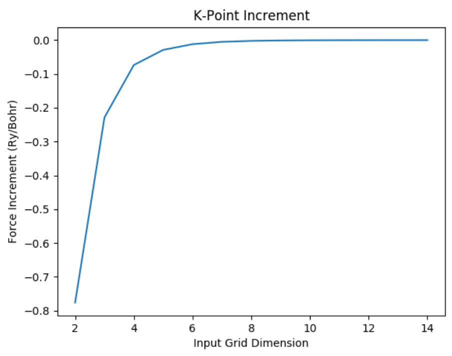
Figure 4: Force vs. Cutoff Energy.

We repeat the same process as in Problem 1 using the same lattice parameter (5 Å), crystal structure, and k-point grid size (4 x 4 x 4), as well as the same range of kinetic energies. In this case, however, we shift the position of the first Ge atom in the lattice by adding 0.05 to the third coordinate of its position through the .positions attribute, corresponding to a fractional shift of 0.05 times the lattice parameter in the z direction. Figure 4 displays the results from these calculations. The force converges to a value of 0.83 Ry/(Bohr-atom) at a value of 30 Ry, in which convergence is defined as above but with a threshold of 10 meV/(Å-atom), or 0.0004 Ry/(Bohr-atom).

4 Problem 4: K-Point Convergence of Forces



(a) Force.

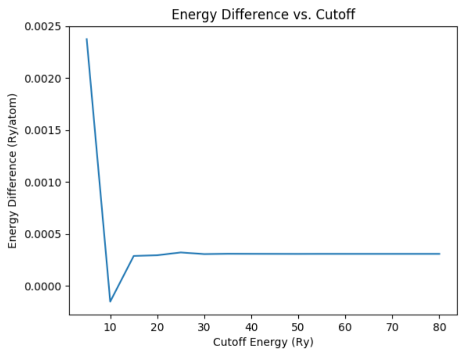


(b) Convergence of Force.

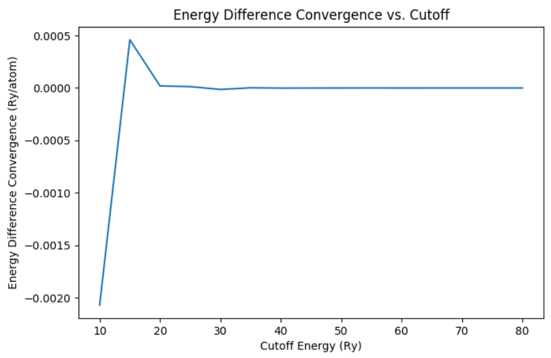
Figure 5: Force vs. Input K-Point Grid Size.

Using the same setup as in Problem 2 (lattice parameter of 5 Åand cutoff energy of 30 Ry) with the same displacement scenario as in the previous problem, we obtain the results shown in Figure 5. Here, the force converges to a value of 0.80 Ry/(Bohr-atom) at a grid size of 11 x 11 x 11, or 216 unique k-points.

5 Problem 5: Cutoff Convergence of Energy Differences



(a) Energy Difference.



(b) Convergence of Energy Difference.

Figure 6: Force vs. Cutoff Energy.

Using a k-point grid of 4 x 4 x 4 and the same lattice parameter as in all previous problems, we perform cutoff convergence as in Problem 1 for two different lattice parameters: 10.70 Bohr (or Å) for one, and 10.75 Bohr (or Å) for the other. Figure 6 shows the calculated energy difference as a function of kinetic energy cutoff between these two cells (the one with the smaller lattice parameter is subtracted from the one with the larger lattice parameter). We see convergence to a final value of 0.0004 Ry/atom at a comparably low kinetic energy cutoff of 20 Ry.

6 Problem 6: Discussion

Based on the above results, we can observe opposite trends in convergence with respect to the kinetic energy cutoff and k-point grid size for absolute energies and forces. For absolute energies, we see a slightly larger kinetic energy cutoff required for convergence (35 Ry) compared to that necessary for force convergence (30 Ry). Conversely, an input grid dimension of 9 is sufficient to allow for convergence of absolute energies with respect to k-points, while a larger input dimension of 11 is necessary for forces. This latter behavior cannot be further contrasted with respect to energy differences, since k-point convergence was not examined. However, we have found that energy differences appear to converge more quickly with respect to the kinetic energy cutoff than either of the other two properties, requiring a cutoff of only 20 Ry. The larger k-point grid requirement with respect to forces seems to make sense, as the force was introduced by displacing the atom from its equilibrium position. As a consequence, some of the symmetries of the crystal may be broken, which could in turn require more k-points to be sampled before convergence. It is possible that the larger kinetic energy cutoff for absolute energies reflects a larger importance of core electrons in computing absolute energies over forces, though the difference between these two cutoffs is fairly small. The significantly lower cutoff energy for the energy differences may be explained by considering potential errors or fluctuations that arise during computation. When taking a difference between two structures, it is possible that

7 Problem 7: Equilibrium Lattice Constant and Bulk Modulus

As we have seen that forces and energy differences appear to show higher cutoff criteria than absolute energies, we increase the size of our calculation to a k-point grid size of 11 and an energy cutoff of 35 Ry.

7.1 A: Lattice Parameter

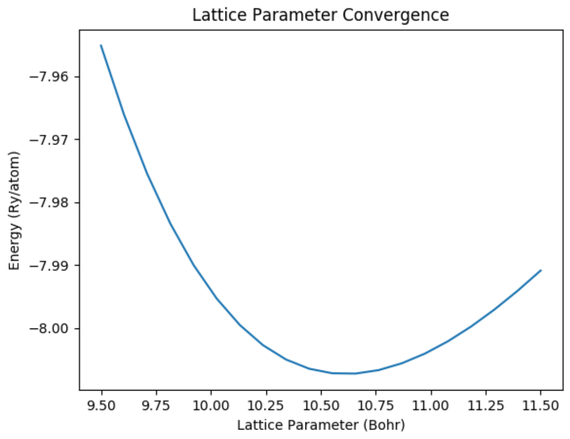


Figure 7: Lattice Parameter Optimization.

We vary the lattice parameter of Ge in the diamond cubic crystal structure between 9.5 and 12.5 Bohr in increments of 0.1, and obtain the results in Figure 7. The k-point grid is set to 11 and the kinetic energy cutoff to 35 to ensure convergence with respect to the previously tested properties. The minimum value of -8.007 Ry/atom is found at 10.61 Bohr, nearly consistent with the experimental results, but off by almost 1%, consistent with the fact that DFT often overbinds systems.

7.2 B: Bulk Modulus

To compute the bulk modulus, we begin with the given relationship:

$$B = -V_0 \frac{\partial P}{\partial V}$$

Here, the proportionality constant is the equilibrium volume of the unit cell, B is the bulk modulus, and P and V are pressure and volume, respectively. Commonly, bulk modulus calculations are solved by invoking a second relationship relating energy and pressure,

$$P = -\frac{\partial E}{\partial V}$$

Combining these results yields the relationship between the bulk modulus and the curvature of the energy-volume relationship:

$$B = V_0 \frac{\partial}{\partial V}\frac{\partial E}{\partial V} = V_0 \frac{\partial^2 E}{\partial V^2}$$

We then need a relation between the volume of the primitive cell and the lattice parameter. A simple empirical relationship can be derived using the output files, or from the geometry of the primitive cell. The volume can be found to be related to the lattice parameter as $V = \frac{a^3}{4}$. With these formulae, we can construct a simple method for obtaining the bulk modulus. We begin by taking the lattice parameter at the minimum energy found above, 10.62 Bohr, as $alat\_min$, and find its corresponding volume, V_min. In order to facilitate unit conversions and input into the lattice scan function, we convert both of these parameters to Åinstead. We approximate the second derivative using a fourth-order expansion of the central difference theorem, such that:

$$f''(x_0) = \frac{-f(x_0 + 2h) + 16f(x_0 + h) - 30f(x_0) + 16f(x_0 - h) - f(x_0 - 2h)}{12h^2}$$

We use a value of h = 1 cubic Angstrom, so that the denominator simply becomes 12, and for each perturbation from the central value x_0, we find the lattice parameter corresponding to the desired volume by inverting the above formula, so $a = (4V)^{1/3}$, and pass this as an input into the DFT code. We obtain the resultant energy from the code in units of electron volts, and obtain the approximation for the second derivative using the above formula. We then multiply by the V_min parameter to yield a final result with units of electron volt per cubic Angstrom. We then convert to units of $J/m^3 = Pa$ for the final result. Using this methodology, we obtain a bulk modulus of 70 GPa, fairly close to the experimental value of 76 GPa. Some of the error in this result may not be due to the simulation itself, but rather a consequence of the finite difference method used to approximate the second derivative. Options to improve this accuracy include using higher-order expansions in deriving the formula or fitting the data (with more points) near the minimum to an equation of state and extracting the value from this equation. With respect to the former option, grids were tested from 0.01 to 1 and from 0.5 to 4 cubic Åto examine the effect of the volume increment parameter on the results, and Figure 8 displays these results. The former grid shows that the values have converged by the initial test of 1, and the latter shows that the method appears to get less accurate for larger values, which is expected as these will be less accurate in approximating the true form of the second derivative, though these fluctuations in the latter case are quite small.

7.3 C: Automatic Optimization

In order to turn on the automatic structure optimization in Quantum Espresso, we change the calculation type in the input file to 'vc-relax' instead of 'scf', which allows the structure to vary during the calculations. We also alter the 'ion dynamics' and 'cell dynamics' parameters in the IONS and CELL subfields of the input file to 'bfgs' to specify the algorithm used for nonlinear optimization. Using an input lattice parameter at the lower bound of the range we tested before (9.5 Bohr), a k-point input dimension of 11, and an energy cutoff of 35 Ry, the output file shows a final lattice parameter of 10.61 Bohr, identical to the value found from the automatic optimization. The energy per atom is equal out to three decimal places at -8.007 Ry/atom as well.

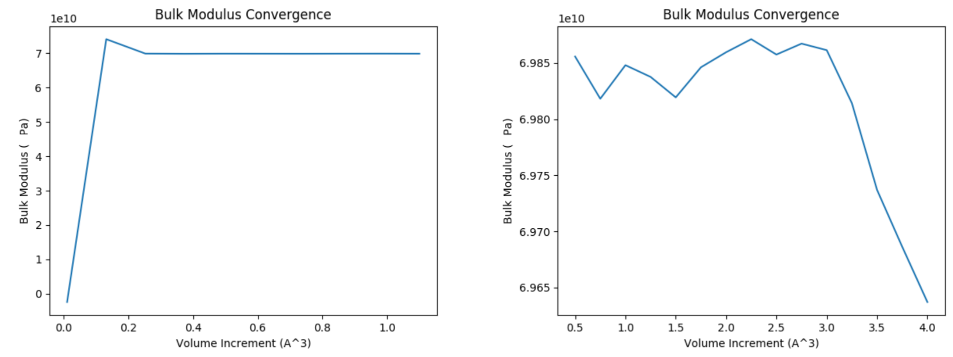


Figure 8: Bulk Modulus Convergence.

8 Bonus 1: Elastic Constants

A cubic crystal can be described by three elastic constants, denoted C_{11}, C_{12}, and C_{44}, which represent normal stress-strain relationship in the same direction, normal strain in response to a stress in a different direction, and shear relationship, respectively. Using a complete unit cell of Ge with diamond cubic structure, we follow the procedure of Mehl et al. (*Intermetallic Compounds: Principles and Pratice*, 1994) in "First principles calculations of elastic properties of metals," who perform volume-conserving transformations on a tetragonal cell to uncover the same elastic constants. However, these strains are performed with respect to the FCC basis vectors, which are defined as:

$$\vec{a_1} = \frac{a_0}{2}\vec{j} + \frac{a_0}{2}\vec{k}$$
$$\vec{a_2} = \frac{a_0}{2}\vec{i} + \frac{a_0}{2}\vec{k}$$
$$\vec{a_3} = \frac{a_0}{2}\vec{i} + \frac{a_0}{2}\vec{j}$$

Here, a_0 is the typical cubic lattice parameter. The function used in Python from ASE to create the structure (crystal) does not appear to create the correct structure if these basis vectors are passed in using the "cell" parameter when the spacegroup is also defined, instead creating a cell with the correct number of atoms but with a volume that is four times too small. Instead, we use the following relations to obtain the three typical orthorhombic vectors ($\vec{a}$, $\vec{b}$, and $\vec{c}$), from the primitive basis vectors:

$$\vec{a} = \vec{a_2} + \vec{a_3} - \vec{a_1}$$
$$\vec{b} = \vec{a_1} + \vec{a_3} - \vec{a_2}$$
$$\vec{c} = \vec{a_1} + \vec{a_2} - \vec{a_3}$$

With the bulk modulus already determined, we need two more constants, which are typically taken as the shear moduli C_{44} and $C_{11} - C_{12}$. The relation $B = \frac{C_{11}+2C_{12}}{3}$ can be used to complete the set and obtain all three elastic constants. To obtain $C_{11} - C_{12}$, we utilize a volume-conserving orthorhombic strain of the form:

$$e_1 = -e_2 = x$$
$$e_3 = x^2/(1 - x^2)$$
$$e_4 = e_5 = e_6 = 0$$

The strain tensor, $\vec{\epsilon}$ is then constructed as:

$$\begin{bmatrix} e_1 & e_6/2 & e_5/2 \\ e_6/2 & e_2 & e_4/2 \\ e_5/2 & e_4/2 & e_3 \end{bmatrix}$$

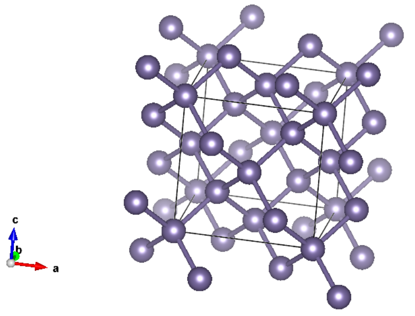


Figure 9: Germanium Unit Cell.

To obtain the new primitive vectors, we perform the matrix multiplication (where $\vec{I}$ is the identity matrix):

$$\begin{bmatrix} \vec{a_1'} \\ \vec{a_2'} \\ \vec{a_3'} \end{bmatrix} = \begin{bmatrix} \vec{a_1} \\ \vec{a_2} \\ \vec{a_3} \end{bmatrix} \cdot [\vec{I} + \vec{\epsilon}]$$

Using this strain field, the volume of the cell remains constant, and the elastic constants can be found through:

$$\Delta E(x) = V(C_{11} - C_{12})x^2$$

We use a similar methodology for obtaining the shear modulus C_{44}, but our strains are instead defined as:

$$\begin{aligned} &e_6 = x \\ &e_3 = x^2/(4 - x^2) \\ &e_1 = e_2 = e_4 = e_5 = 0 \end{aligned}$$

This yields an energy difference of:

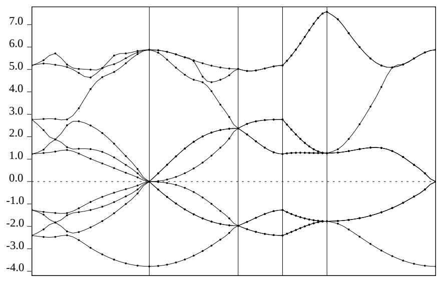
$$\Delta E(x) = \frac{Vx^2}{2} C_{44}$$

We implement these strain fields in two different runs of the Ge script 2 python code. In the first of the two runs, the strain field for C_{44} is used. Two energies are calculated- one corresponding to the normal lattice, and one to the strained lattice. The normal lattice is displayed above in Figure 9. Note that, for all calculations in this problem, the "nosym" parameter in the &SYSTEM input field is set to True, so no symmetries are assumed which could disrupt computation of the strained cell. The energy difference obtained through this calculation is found to be 0.00468 eV. Inserting this into the formula above, with a strain of 0.01 and a cell volume of 176.55 cubic Å, we obtain $C_{44} = 84GPa$. A similar procedure is used for the other strain field, for which we obtain an energy difference of 0.00898 eV. Using the same cell volume and strain, this yields $C_{11} - C_{12} = 81GPa$. Invoking our previously obtained bulk modulus of 70 GPa and the above formula for bulk modulus as a function of the two elastic constants, and solving the two equations simultaneously, we obtain $C_{12} = 43GPa$ and $C_{11} = 124GPa$. Comparing our results with the literature, we consider the work of Bond et al. (*Physical Review*, 1950) "The Elastic Constants of Germanium Single Crystals." In this work, the elastic constants are obtained as $C_{11} = 130GPa$, $C_{12} = 49GPa$, and $C_{44} = 67GPa$. Of our results, the only one that differs substantially is C_{44}. Given that we only test one strain value (due to the high computational cost of running a calculation with eight atoms without any symmetry, a cutoff of 35 Ry, and an 11 x 11 x 11 k-point grid), it is possible that this value had not converged with respect to the strain, and this may one source of discrepancy. It is also worth noting that use of the experimental bulk modulus value, 76 GPa, yields values for the other two elastic constants that exactly match the above cited experimental results.

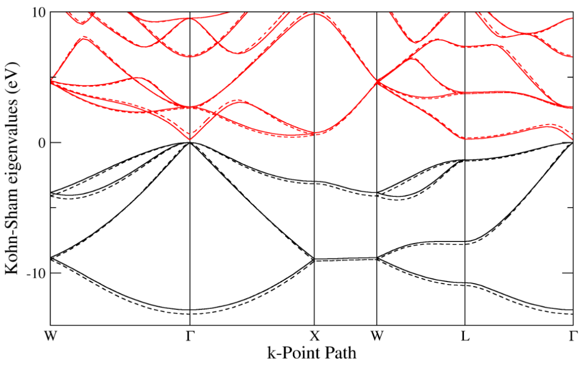
9 Bonus 2: Band Structure

To uncover the band structure of Ge, we change the calculation parameter to *bands* and specify a list of k-points along high symmetry directions. These directions are chosen as those specified in the ASE documentation, which are in turn drawn from Setyawana and Curtarolo (*Computational Materials Science*, 2010). In the Python script, these can be obtained using the get special points function from ase.dft.kpoints. The k-point path is chosen to align with that used in the literature ($W - \Gamma - X - W - L - \Gamma$). We also specify the nbnd parameter to be 10 to account for the valence and first four conduction bands in the &$SYSTEM$ input field. The parameter disk io is changed to 'low' instead of 'none,' which causes creation of wavefunction and energy density files along with the typical input and output files. Since multiple calculations from pw.x are necessary, the command line interface is used in lieu of a Python script. Prior to changing the calculation type, the typical 'scf' calculation is run first in the same folder, and then a 'bands' calculation is run using the same directory. On the command line, pw.x < pwscf.in > pwscf.out runs the first of these calculations, and pw.x < pwscfbnd.in runs the second. The bands.x function is then used to sort the bands using the input file bands.in, and finally plotband.x is used to plot the results. The Fermi level was found to be -3.3 eV by shifting the value around until it corresponded with the approximate location of the band gap. The input files are included in the appendix below.

Figure 10 displays the result of the DFT band structure calculation as well as results from a thesis, *Development of Non-Local Density Functional Methods* by Dominik Bogdan Jochym at the University of Durham. Qualitatively, the results are somewhat similar, with the QE results showing more fluctuations in the individual paths. Importantly, the QE results do not show a band gap, while a small one is present in the thesis results. Therefore, using this method would incorrectly predict that germanium is metallic rather than a semiconductor.



(a) DFT.



(b) Experimental.

Figure 10: Germanium Band Structure.

10 Appendix

Attached below is the source code for the modified Ge script file. The major modifications are in the main function, where the Q parameter specifies which sub-region to enter. These values correspond to the question to be answered (and Q = 8 is simply used to run any value for the automatic optimzation in part 7C). The displacement value in the make struc function is currently set to 0.00, and can be adjusted to 0.05 for problems 3 and 4, to add forces. The runpath in compute energy can also be changed to save to different folderrs depending on the question. An updated version of pwscf.py was also constructed that extracts the walltime in addition to energy, force, stress, and pressure to obtain the calculation times. The added code in lattice scan parses the obtained string to extract only the numbers and decimal point.

```python
    from my_labutil.src.plugins.pwscf import *
from ase.spacegroup import crystal
from ase.build import *
from ase.io import write
import matplotlib.pyplot as plt
import numpy as np


def make_struc(alat):
    """
    Creates the crystal structure using ASE.
    :param alat: Lattice parameter in angstrom
    :return: structure object converted from ase
    """
    # set primitive_cell=False if you want to create a simple cubic unit cell with 8 atoms
    gecell = crystal('Ge', [(0, 0, 0)], spacegroup=227, cellpar=[alat, alat, alat, 90, 90,
    90], primitive_cell=True)
    gecell.positions[0][2]=gecell.positions[0][2]+0.00
    # change above to +0.05 for numbers 3 and 4
    # check how your cell looks like
    write('s.cif', gecell)
    structure = Struc(ase2struc(gecell))
    return structure


def compute_energy(alat, nk, ecut):
    """
    Make an input template and select potential and structure, and the path where to run
    """
    potname = 'Ge.pz-bhs.UPF'
    pseudopath = os.environ['ESPRESSO_PSEUDO']
    potpath = os.path.join(pseudopath, potname)
    pseudopots = {'Ge': PseudoPotential(name=potname, path=potpath, ptype='uspp', element='
    Ge', functional='LDA')}
    struc = make_struc(alat=alat)
    kpts = Kpoints(gridsize=[nk, nk, nk], option='automatic', offset=False)
    runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab2/Problem7", str(ecut)+'.'+
    str(nk)+'.'+str(alat)))
    input_params = PWscf_inparam({
        'CONTROL': {
            'calculation': 'scf',
            'pseudo_dir': pseudopath,
            'outdir': runpath.path,
            'tstress': True,
            'tprnfor': True,
            'disk_io': 'none',
        },
        'SYSTEM': {
            'ecutwfc': ecut,
             },
        'ELECTRONS': {
            'diagonalization': 'david',
            'mixing_beta': 0.5,
            'conv_thr': 1e-7,
        },
        'IONS': {

        },
        'CELL': {

        },

        })

    output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
                               params=input_params, kpoints=kpts)
    output = parse_qe_pwscf_output(outfile=output_file)
    return output

def lattice_scan(nk, ecut, alat):
    #nk = 3
    #ecut = 30
    #alat = 5.0
    output = compute_energy(alat=alat, ecut=ecut, nk=nk)
    energy = output['energy']
    kpts = output['kpoints']
    forces = output['force']
    walltime = output['walltime']
    wallt = []
    for t in walltime:
        if str.isdigit(t) or '.' in t:
            wallt.append(str(t))
    #print(energy)
    #print(wallt)
    return energy, float(''.join(wallt)), kpts, forces


if __name__ == '__main__':
    # put here the function that you actually want to run
    Q = 7

    if Q == 1:
        nk = 4
        energy=[]
        ediff = []
        caltime = []
        forces = []
        alat = 5.0
        for ecut in range(5,85,5):
            energy.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[0]/(2*13.605698066))
            caltime.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[1])
            forces.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[3])
        for en in range(0,len(energy)-1):
            ediff.append(energy[len(energy)-1]-energy[en])

        plt.plot(range(5, 85, 5), energy)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Energy (Ry/atom)')
        axes = plt.gca()
        #axes.set_ylim([0, 0.2])
        plt.title("Cutoff Convergence")
        plt.show()

        plt.plot(range(10, 85, 5), ediff)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Energy Difference (Ry/atom)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Convergence Energy Change")
        plt.show()

        plt.plot(range(5, 85, 5), caltime)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Calculation Time (s)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Calculation Time")
        plt.show()

    elif Q == 2:
        ecut = 30
        energy = []
        ediff = []
        caltime = []
        kpts=[]
        alat=5.0
        for nk in range(1, 13):
            energy.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[0] / (2 * 13.605698066))
            caltime.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[1])
            kpts.append(lattice_scan(nk=nk, ecut=ecut, alat=alat)[2])
        for en in range(0, len(energy) - 1):
            ediff.append(energy[len(energy)-1] - energy[en])
        plt.plot(kpts, energy)
        plt.xlabel('Unique K-Points')
        plt.ylabel('Energy (Ry/atom)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("K-Point Convergence")
        plt.show()

        plt.plot(kpts[1:len(kpts)], ediff)
        plt.xlabel('Unique K-Points')
        plt.ylabel('Convergence Energy Difference (Ry)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Convergence Energy Change")
        plt.show()

        plt.plot(kpts, caltime)
        plt.xlabel('Unique K-Points')
        plt.ylabel('Calculation Time (s)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Calculation Time")
        plt.show()

        plt.plot(range(1,13), energy)
        plt.xlabel('Input K-Points')
        plt.ylabel('Energy (Ry/atom)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("K-Point Convergence")
        plt.show()

        plt.plot(range(2,13), ediff)
        plt.xlabel('Input K-Points')
        plt.ylabel('Convergence Energy Difference (Ry)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Convergence Energy Change")
        plt.show()

        plt.plot(range(1,13), caltime)
        plt.xlabel('Input K-Points')
        plt.ylabel('Calculation Time (s)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Calculation Time")
        plt.show()

        plt.plot(range(1,13), kpts)
        plt.xlabel('Input K-Points')
        plt.ylabel('Unique K-Points')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("K-Points")
        plt.show()

    elif Q == 3:
        forces=[]
        fdiff=[]
        alat=5.0
        for ecut in range(5, 85, 5):
            forces.append(lattice_scan(nk=4, ecut=ecut, alat=alat)[3]/2)
            print(forces)

        for en in range(0,len(forces)-1):
            fdiff.append(forces[len(forces)-1]-forces[en])

        plt.plot(range(5, 85, 5), forces)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Force (Ry/Bohr)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Cutoff Convergence")
        plt.show()

        plt.plot(range(10, 85, 5), fdiff)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Force Increment (Ry/Bohr)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("Cutoff Increment")
        plt.show()

    elif Q == 4:
        forces = []
        fdiff = []
        alat=5.0
        for nk in range(1,15):
            forces.append(lattice_scan(nk=nk, ecut=30, alat=alat)[3]/2)
            print(forces)

        for en in range(0, len(forces) - 1):
            fdiff.append(forces[len(forces)-1] - forces[en])

        plt.plot(range(1,15), forces)
        plt.xlabel('Input Grid Dimension')
        plt.ylabel('Force (Ry/Bohr)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("K-Point Convergence")
        plt.show()

        plt.plot(range(2,15), fdiff)
        plt.xlabel('Input Grid Dimension')
        plt.ylabel('Force Increment (Ry/Bohr)')
        axes = plt.gca()
        # axes.set_ylim([0, 0.2])
        plt.title("K-Point Increment")
        plt.show()

    elif Q == 5:
        e1=[]
        e2=[]
        alat1=(10.70*0.529177249)
        alat2=(10.75*0.529177249)
        ediff=[]
        ediff2=[]
        nk=4
        for ecut in range(5,85,5):
            e1temp=lattice_scan(nk=nk,ecut=ecut,alat=alat1)[0]/(2 * 13.605698066)
            e1.append(e1temp)
            e2temp=lattice_scan(nk=nk,ecut=ecut,alat=alat2)[0]/(2 * 13.605698066)
            e2.append(e2temp)
            ediff.append(e2temp-e1temp)

        for en in range(0, len(ediff) - 1):
            ediff2.append(ediff[len(ediff)-1] - ediff[en])

        plt.plot(range(5,85,5),ediff)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Energy Difference (Ry/atom)')
        axes = plt.gca()
        plt.title('Energy Difference vs. Cutoff')
        plt.show()

        plt.plot(range(10, 85, 5), ediff2)
        plt.xlabel('Cutoff Energy (Ry)')
        plt.ylabel('Energy Difference Convergence (Ry/atom)')
        axes = plt.gca()
        plt.title('Energy Difference Convergence vs. Cutoff')
        plt.show()

    elif Q == 7:
        nk=11
        ecut=35
        energy=[]
        for alat in np.linspace(10,11.1,10):
            alat=alat*0.529177249
            energy.append(lattice_scan(nk=nk,ecut=ecut,alat=alat)[0]/(2 * 13.605698066))

        plt.plot(np.linspace(10,11.1,10),energy)
        plt.xlabel('Lattice Parameter (Bohr)')
        plt.ylabel('Energy (Ry/atom)')
        plt.title('Lattice Parameter Convergence')
        plt.show()

        B=[]
        for v in np.linspace(0.01,1.1,10):
            ind_min = np.argmin(energy)
            alat_min=0.529177249*np.linspace(10,11.1,10)[ind_min]
            V_min=alat_min**3/(4)
            alatf2=(4*(V_min+2*v))**(1/3)
            alatf1=(4*(V_min+v))**(1/3)
            alatfn1=(4*(V_min-v))**(1/3)
            alatfn2=(4*(V_min-2*v))**(1/3)
            f2=lattice_scan(nk=nk,ecut=ecut,alat=alatf2)[0]
            f1=lattice_scan(nk=nk,ecut=ecut,alat=alatf1)[0]
            f0=(2 * 13.605698066)*energy[ind_min]
            fn1=lattice_scan(nk=nk,ecut=ecut,alat=alatfn1)[0]
            fn2=lattice_scan(nk=nk,ecut=ecut,alat=alatfn2)[0]

            d2EdV2=(-f2+16*f1-30*f0+16*fn1-fn2)/(12*v**2)
            B.append(V_min*((10**(10))**3*(1.6*10**(-19)))*d2EdV2)
            print(alat_min)
            print(V_min)
            print(f2)
            print(f1)
            print(f0)
            print(fn1)
            print(fn2)
            print(B)

        plt.plot(np.linspace(0.01,1.1,10), B)
        plt.xlabel('Volume Increment (A^3)')
        plt.ylabel('Bulk Modulus (GPa)')
        plt.title('Bulk Modulus Convergence')
        plt.show()

    elif Q == 8:
        lattice_scan(11,35,5.0)

```

Included below is a modified version of the above script to be used with the bonus questions. Here, a number of notable changes are made for each question. For the first bonus, the input to the crystal function is changed to a 3 x 3 matrix instead of the [alat,alat,alat,90,90,90] form used previously, in order to specify the strained lattice vectors properly. Each function from lattice scan to compute energy, to make struc now accepts the matrix a, which contains these vectors, in addition to the lattice parameter alat. For this question, the primitive argument is also set to False to obtain the complete unit cell, and the nosym parameter in &SYSTEM is set to True so no symmetries are assumed. Both strain fields are present, with the current state being that used to obtain the quantity $C_{11} - C_{12}$. The commented version can be pasted in for C_{44}. The matrix a contains the unstrained lattice vectors, and the lines using at are simply there to ensure that the transformation from primitive basis vectors to unit cell vectors reconstructs the correct matrix. The an matrix performs the same transformation to obtain the strained unit cell vectors from the strained basis vectors.

```
  from my_labutil.src.plugins.pwscf import *
from ase.spacegroup import crystal
from ase.build import *
from ase.io import write
import matplotlib.pyplot as plt
import numpy as np
import ase.dft.kpoints as k


def make_struc(alat, a):
    """
    Creates the crystal structure using ASE.
    :param alat: Lattice parameter in angstrom
    :return: structure object converted from ase
    """
    # set primitive_cell=False if you want to create a simple cubic unit cell with 8 atoms
    gecell = crystal('Ge', [(0, 0, 0)], spacegroup=227, cell=a, primitive_cell=False)
     #check how your cell looks like
    gecell.positions[0][2]=gecell.positions[0][2]+0.00
    ge = bulk('Ge', 'diamond', a=5.61)
    points = k.get_special_points('fcc', ge.cell)
    print(points)
    write('s2.cif', gecell)
    structure = Struc(ase2struc(gecell))
    return structure


def compute_energy(alat, nk, ecut, a):
    """
    Make an input template and select potential and structure, and the path where to run
    """
    potname = 'Ge.pz-bhs.UPF'
    pseudopath = os.environ['ESPRESSO_PSEUDO']
    potpath = os.path.join(pseudopath, potname)
    pseudopots = {'Ge': PseudoPotential(name=potname, path=potpath, ptype='uspp', element='
    Ge', functional='LDA')}
    struc = make_struc(alat=alat, a=a)
    kpts = Kpoints
    (6,[[0,0,0,1],[0.375,0.375,0.75,1],[0.5,0.5,0.5,1],[0.625,0.625,0.625,1],[0.5,0.25,0.75,1],[0.5,0,0.5,
     option='crystal_b', offset=False)
    print(kpts)
    runpath = Dir(path=os.path.join(os.environ['WORKDIR'], "Lab2/Problem9/", str(ecut)+'.'+
    str(nk)+'.'+str(alat)))
    input_params = PWscf_inparam({
        'CONTROL': {
            'calculation': 'bands',
            'pseudo_dir': pseudopath,
            'outdir': runpath.path,
            'tstress': True,
            'tprnfor': True,
            'disk_io': 'low',
        },
        'SYSTEM': {
            'ecutwfc': ecut,
            'nosym': False,
            'nbnd': 5
             },
        'ELECTRONS': {
            'diagonalization': 'david',
            'mixing_beta': 0.5,
            'conv_thr': 1e-7,
        },
        'IONS': {

        },
        'CELL': {

        },
        'K_POINTS': {
                        '{crystal_b}'
                        '6'
                        '0 0 0 1'
                        '0.375 0.375 0.75 1'
                        '0.5 0.5 0.5 1'
                        '0.625 0.25 0.625 1'
                        '0.5 0.25 0.75 1'
                        '0.5 0 0.5 1'
                    }
        })


    output_file = run_qe_pwscf(runpath=runpath, struc=struc,  pseudopots=pseudopots,
                               params=input_params, kpoints=kpts)
    output = parse_qe_pwscf_output(outfile=output_file)
    return output



def lattice_scan(nk, ecut, alat, a):
    #nk = 3
    #ecut = 30
    #alat = 5.0
    output = compute_energy(alat=alat, ecut=ecut, nk=nk, a=a)
    energy = output['energy']
    kpts = output['kpoints']
    forces = output['force']
    walltime = output['walltime']
    wallt = []
    if isinstance(walltime, int):
        wallout=walltime
    else:
        for t in walltime:
            if str.isdigit(t) or '.' in t:
                wallt.append(str(t))
        wallout=float(''.join(wallt))
    #print(energy)
    #print(wallt)
    return energy, wallout, kpts, forces


if __name__ == '__main__':
    # put here the function that you actually want to run
    Q = 9

    if Q == 8:
        x=0.01
        alat=5.61
        e1=x
        e2=-x
        e3=x**2/(1-x**2)
        e4=0
        e5=0
        e6=0
        e=np.array([[e1,e6/2,e5/2],[e6/2,e2,e4/2],[e5/2,e4/2,e3]])
        a1=[0,alat,alat]
        a2=[alat,0,alat]
        a3=[alat,alat,0]
        a=np.array([a1,a2,a3])
        a=a/2
        print(a)
        a2=a
        at1=a[1]+a[2]-a[0]
        at2=a[2]+a[0]-a[1]
        at3=a[0]+a[1]-a[2]
        a=np.array([at1,at2,at3])
        e=e+np.identity(3)
        print(e)
        anew=np.dot(a2,e)
        print(anew)
        an1=anew[1]+anew[2]-anew[0]
        an2=anew[2]+anew[0]-anew[1]
        an3=anew[0]+anew[1]-anew[2]
        an=np.array([an1,an2,an3])

        print(a)
        print(an)
        E1=lattice_scan(nk=11,ecut=35,alat=5.61, a=a)[0]
        E2=lattice_scan(nk=11,ecut=35,alat=5.61, a=an)[0]
        print(E1)
        print(E2)

        '''
        For C44 instead,

        e1=0
        e2=0
        e4=0
        e5=0
        e6=x
        e3=(x)**2/(4-x**2)
        '''

    elif Q == 9:
        a=1
        lattice_scan(nk=11,ecut=50,alat=5.61, a=a)

```

For the final question, not all of the code is included below. Instead, the input files for pw.x and bands.x are provided. Initially, an scf calculation is run, using only the primitive lattice in the same setup as the previous (non-bonus) questions with no displacement. Once this is complete, the calculation is changed to 'bands', the 'nbnd' parameter in &SYSTEM is set to 5, the 'disk_io' parameter in &CONTROL is set to 'low' so the necessary wavefunction and pwscf.save files are created, and the K_POINTS input is added below CELL to specify the special k-point directions sampled in the band structure calculation. The first section is used for the typical scf calculation with a cutoff of 35 Ry and an 11 x 11 x 11 grid.

```
&CONTROL
    disk_io = 'low'
    outdir = '/home/bond/WORK/Lab2/Problem9/35.11.5.61'
    tprnfor = .true.
    tstress = .true.
    calculation = 'scf'
    pseudo_dir = '/home/bond/Software/qe-6.0/pseudo'
/
&SYSTEM
    ecutwfc = 35
    nat = 8
    nosym = .false.
    ntyp = 1
    ibrav = 0
/
&ELECTRONS
    conv_thr = 1e-07
    diagonalization = 'david'
    mixing_beta = 0.5
/
&IONS
/
&CELL
/
K_POINTS {automatic}
 11 11 11  0 0 0
ATOMIC_SPECIES
  Ge 72.61 Ge.pz-bhs.UPF
CELL_PARAMETERS {angstrom}
 5.61 0.0 0.0
 3.435134271608326e-16 5.61 0.0
 3.435134271608326e-16 3.435134271608326e-16 5.61
ATOMIC_POSITIONS {angstrom}
  Ge 0.00000 0.00000 0.00000
  Ge 0.00000 2.80500 2.80500
  Ge 2.80500 2.80500 0.00000
  Ge 2.80500 0.00000 2.80500
  Ge 4.20750 1.40250 4.20750
  Ge 1.40250 1.40250 1.40250
  Ge 1.40250 4.20750 4.20750
  Ge 4.20750 4.20750 1.40250
```

The second input file is used for the band structure calculation with 10 bands and the previously specified k-point path in reciprocal space. The weights next to each k-point were chosen empirically so that the number of points along each section of the path were similar.

```
&CONTROL
    disk_io = 'low'
    outdir = '/home/bond/WORK/Lab2/Problem9/35.11.5.61'
    tprnfor = .true.
    tstress = .true.
    calculation = 'bands'
    pseudo_dir = '/home/bond/Software/qe-6.0/pseudo'
/
&SYSTEM
    ecutwfc = 35
    nat = 8
    nosym = .false.
    ntyp = 1
    nbnd = 10
    ibrav = 0
/
&ELECTRONS
    conv_thr = 1e-07
    diagonalization = 'david'
    mixing_beta = 0.5
/
&IONS
/
&CELL
/
K_POINTS crystal_b
    6
```

```
     0.5     0.25     0.75     10.0
     0.0000000000     0.0000000000     0.0000000000     10.0
     0.5    0.00    0.5     5.0
     0.5     0.25     0.75     10.0
     0.5     0.5     0.5     10.0
     0.0     0.0     0.0     10.0
ATOMIC_SPECIES
  Ge 72.61 Ge.pz-bhs.UPF
CELL_PARAMETERS {angstrom}
 5.61 0.0 0.0
 3.435134271608326e-16 5.61 0.0
 3.435134271608326e-16 3.435134271608326e-16 5.61
ATOMIC_POSITIONS {angstrom}
  Ge 0.00000 0.00000 0.00000
  Ge 0.00000 2.80500 2.80500
  Ge 2.80500 2.80500 0.00000
  Ge 2.80500 0.00000 2.80500
  Ge 4.20750 1.40250 4.20750
  Ge 1.40250 1.40250 1.40250
  Ge 1.40250 4.20750 4.20750
  Ge 4.20750 4.20750 1.40250
```

Finally, the bands.x input file specifies the directory, filename, output filename, and a parameter which prevents crossovers during plotting.

```
&BANDS
     outdir = '/home/bond/WORK/Lab2/Problem9/35.11.5.61/',
     prefix = 'pwscf',
     filband = 'Ge_bands.dat',
     no_overlap= .true.
/
```