Schroedinger–Poisson solver for 2D materials: documentation

Augustin Bussy Giovanni Pizzi Marco Gibertini

May 2, 2017

1 Introduction

This code aims at self-consistently solving the coupled Schroedinger–Poisson equations in 2D materials. More precisely, in the current implementation it is designed to simulate nanosheets of a single material with regions of different strains. The Schroedinger equation is in 1D, meaning that the strain can vary on a single axis (x), as illustrated in Figure 1. Moreover, the material is assumed to be infinite in the y-direction and periodic in the x-direction.

Two types of calculations can be made: *single-point* or *map*. The former (*single-point* simulates a specific setup described in an input file (see Sec. 3) to produce a band profile and/or a carrier density profile. This is a useful tool for experiments as it allows to simulate any strain profile. The *map* calculation type, instead, screens over different setups of strained/unstrained material to investigate, e.g., insulator-to-conductor phase transitions.

This code uses a multi-scale approach, meaning that the materials properties must be provided as input, and typically they come from DFT calculations. These quantities include effective masses and polarization charges for different strains. The properties for at least 4 or 5 strains should be provided for accurate fitting. It is also not recommended to extrapolate strains larger than those coming from DFT, as the trends could change.

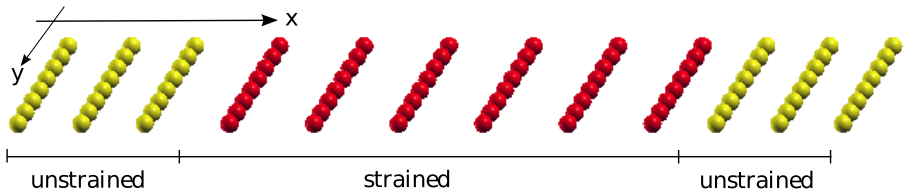

Figure 1: Example of simulation setup: Periodic alternation of strained and unstrained regions of a given material.

2 Compiling and running up the code

The code is distributed in the `code` folder, and consists of three main files. The bulk of the numerical computations is written in Fortran in the `schrpoisson_wire.f90` file. This is interfaced to python using `f2py`, and the interface has been generated by us in the `schrpoisson_wire.pyf`

file. (If you decide to modify the name, number or signature of the Fortran functions, you need to regenerate this file. This is possible by running the command commented in the Makefile.)

To compile the code, run the `make` command in the `code` folder. You will need to have the `f2py` executable[1], that is part of `numpy`.

After running `make`, the main python code (`2Dschrpoisson.py`) is ready to be run using the following command:

```
python 2Dschrpoisson.py {material_properties}.json {calc_input}.json
```

Note that, if the two json files are not in the same folder, you need to specify the full (relative or absolute) path to them.

3 Input

Two input files are necessary to every calculation: one containing the materials properties and the other the calculation setup.

3.1 Material Properties File

The properties of the material at different strains must be provided in `json` file as first command line option. The file should contain a dictionary, where the key is a string representation of the float value, and the value should be a dictionary with a given number of keys, as shown in the example below.

The following example is taken from the `SnSe_example.json` file in the `input_example` folder. Note that the properties of the unstrained material are *mandatory* and must be provided under the "`0.00`" key. Below the example, there are some explanatory comments.

File example

```
{"0.00": {
    "alpha_xx": 10.22,
    "x_lat": 4.408,
    "y_lat": 4.288,
    "polarization_charge": 0.0,
    "vacuum_level": 3.471695,
    "valence_gamma": {
      "energy": -1.508255819,
      "conf_mass": 1.755925079,
      "DOS_mass": 2.7330924,
      "degeneracy": 1
    },
    "valence_gamma-X": {
      "energy": -0.8832657543,
      "conf_mass": 0.125168454,
      "DOS_mass": 0.159070572,
      "degeneracy": 2
    },
```

[1] In most cases, if you already have a recent python distribution and `pip`, you just need to run `pip install numpy`, or use your package manager to install python-numpy. See here for more information: `https://docs.scipy.org/doc/numpy-dev/f2py/`

```
    "valence_gamma-Y": {
      "energy": -1.064317364,
      "conf_mass": 0.109554071,
      "DOS_mass": 0.159511425,
      "degeneracy": 2
    },
    "conduction_gamma": {
      "energy": 0.6090962485,
      "conf_mass": 2.741100107,
      "DOS_mass": 2.994158008,
      "degeneracy": 1
    },
    "conduction_gamma-X": {
      "energy": 0.0902128713,
      "conf_mass": 0.110807373,
      "DOS_mass": 0.190387132,
      "degeneracy": 2
    },
    "conduction_gamma-Y": {
      "energy": 0.05720314215,
      "conf_mass": 0.131542031,
      "DOS_mass": 0.130378261,
      "degeneracy": 2
    }
  },
  "0.01": {
    ...
  },
  ...
}
```


Description of the keys in each subdictionary, for each strain

`alpha_xx` Value of the polarizability of the strained material in Å. x is the direction along which different strains/materials alternate (see Fig. 1).

`x_lat`, `y_lat` The lattice parameters of the unit cell in Å. *Only needed for the unstrained material.*

`polarization_charge` Polarization charge at the edge of the material in units of electron per unit cell width, *i.e.* e/b.

`vacuum_level` Needed to align energy levels across the setup (in eV).

Energy extrema keys They need to either contain the string `valence` or `conduction` (this is a requirement by the code). The assigned value is a subdictionary with the energy of the band extremum in eV (`energy`), the confinement mass (`conf_mass`) and density of states (DOS) mass (`DOS_mass`), that are respectively the effective masses in the x and y directions), and the degeneracy of the energy level in the first BZ (`degeneracy`).

3.2 Calculation Input File

Each type of calculation (single-point or map) has its own input file template with different keys. In both cases, the input file should be in `json` format and be provided as the second command line argument.

The following examples can be found in the `input_examples` folder. In the same folder, a simple python script (`create_calc_input.py`) could be useful to write a json file starting from a python dictionary.

3.2.1 Single-point calculation

```
{
  "calculation": "single_point",
  "out_dir": "single_point_output",
  "smearing": true,
  "KbT": 0.005,
  "max_iteration": 1000,
  "plot_fit": false,
  "nb_of_states_per_band": 2,
  "delta_x": 0.5,
  "setup": {
    "slab1": {
      "polarization": "positive",
      "strain": 0.0,
      "width": 15.0
    },
    "slab2": {
      "polarization": "positive",
      "strain": 0.08,
      "width": 30.0
    },
    "slab3": {
      "polarization": "positive",
      "strain": 0.0,
      "width": 15.0
    }
  }
}
```

Description of the dictionary keys

`calculation` The value `"single_point"` indicates the type of calculation

`out_dir` The name of the folder where the output data is dumped

`smearing` If `true`, a Marzari–Vanderbilt smearing is imposed for the electron distribution. It helps for the convergence of the self-consistent Schroedinger–Poisson loop and allows better comparisons with DFT calculations

`KbT` In case of smearing, this is the imposed electronic temperature (Ry)

`max_iteration` The number of allowed steps for the self-consistent Schroedinger–Poisson loop. 1000 is a safe number

`plot_fit` If `true`, the fitting of materials properties for any strain is displayed interactively in a series of graphs using matplotlib (that needs to be installed)

`nb_of_states_per_band` Useful to limit the size of the output data. In this example, only two quantum states per band (6 in total) will be reported

`delta_x` The size of the discretization step in space (Å)

`setup` The value is a subdictionary with the description of each slab forming the setup. For each layer, one has to specify the `polarization` (`positive` or `negative`: positive if holes accumulate towards larger x), the `strain` of the material and the `width` of the slab (in Å). There is no limit to the numbers of layers, but the keys must follow the syntax `slab%d` where `%d` should be replaced with the integer, starting from 1. It is not recommended to include strains higher than the largest one in the material properties file.

Note: it is also *mandatory* that the first and the last slabs have identical strain, since the code does not cope with polarization charges at the edges of the cell.

3.2.2 Map calculation

```
{
  "calculation": "map",
  "out_dir": "map_output",
  "smearing": true,
  "KbT": 0.005,
  "max_iteration": 1000,
  "plot_fit": false,
  "nb_of_steps": 200,
  "upper_delta_x_limit": 0.5,
  "strain": {
    "max_strain": 0.1,
    "min_strain": 0.0,
    "strain_step": 0.01
  },
  "width": {
    "min_width": 10.0,
    "width_step": 5.0,
    "max_width": 60.0
  }
}
```

Description of the dictionary keys

`calculation` The value `"map"` indicates the type of calculation

`out_dir` The name of the folder where the output data is dumped

`smearing` If `true`, a Marzari–Vanderbilt smearing is imposed for the electron distribution. It helps for the convergence of the self-consistent Schroedinger–Poisson loop and allows better comparisons with DFT calculations

`KbT` In case of smearing, this is the imposed electronic temperature (Ry)

`max_iteration` The number of allowed steps for the self-consistent Schroedinger–Poisson loop. 1000 is a safe number

`plot_fit` If `true`, the fitting of materials properties for any strain is displayed in a series of graphs using matplotlib (that needs to be installed)

`nb_of_steps` Since a wide range of setup sizes is spanned in a map calculation, a common step size might be a problem. It is instead dynamically adapted as

$$\Delta x = \frac{L_{tot}}{\texttt{nb\_of\_steps}}$$

`upper_delta_x_limit` In case an (upper) limit should be imposed on the step size:

$$\Delta x = min\left(\frac{L_{tot}}{\texttt{nb\_of\_steps}}, \texttt{upper\_delta\_x\_limit}\right)$$

`strain` The assigned value is a subdictionary with the lower and upper bound (`min_strain` and `max_strain`) of the strains over which calculations will be done, as well as the step between strains `strain_step`

`width` The assigned value is a subdictionary with the lower and upper bound (`min_width` and `max_width`) of the width over which calculations will be done, as well as the step between widths `width_step`.

Note: the width refers to the unstrained material and the size of the strained slab is computed as $L_{strained} = (1 + \epsilon)L_{unstrained}$.

In this example, 121 single-point calculations will be conducted with every combination of strain $\epsilon = 0.0, 0.01, \ldots, 0.1$ and width $L = 10, 15, \ldots, 60$.

4 Output

4.1 Single-Point Calculation

In single-point calculations, three output files are generated. The first one (`general_info.txt`) contains general output information such as the number of iterations needed, convergence parameters and the total free carrier density. The second one (`band_data.txt`) contains all the necessary information to produce a band profile (Fermi energy, potential profile of each band, their respective quantum states, etc.). Finally, the last file (`density_profile.txt`) contains the free-electron and free-hole density as a function of position.

4.2 Map Calculation

Map calculations produce a single output file in which, among others, the total electron density is given for each strain/width combination. This can be then used to produce a two-dimensional color map.