Example: pre-processing and self-consistent potential energy calculation

This file is intended to detail the calculation presented in Sections 4.1 and 4.2 of our *BinPo* manuscript. Importantly, we remind you that any *BinPo* component looks for the input parameters at command-line at first, whereas the omitted and more advanced parameters are set by default from the configuration file. Keeping this in mind, we will show how the configuration files are set for each case and what to type from command-line. For a description of all the parameters in the configuration files, see *~/config_files/help_config.md*. Those parameters updatable by command-line in any *BinPo* components (*BP-component.py*) can be checked by means of help command as:

$ python BP-component.py -h

Example 4.1: Pre-processing step with *BP-preproc.py*

First of all, let's check the help for *BP-preproc.py* component from the command-line. So, open a terminal inside the *BinPo* folder and type:

$ python BP-preproc.py -h

We will get the needed information about the component usage. There is no configuration file associated to this component. That's because it needs just two mandatory arguments, the material (*mt*) and the crystal face or confinement direction (*cfd*). The remaining parameters are extracted internally from the *BPdatabase.py* module. We will now create a folder holding the slab Hamiltonian elements for a given material and along a specific confinement direction. In this example, STO and the (100) direction are selected by typing:

$ python BP-preproc.py -mt STO -cfd 100

The output text (which is not logged in pre-processing steps) will show the input parameters data and the main details of the DFT original calculations for the Wannier file loaded. The latter information is taken from Python dictionaries inside the *BPdatabase.py* module.

This run could take some minutes depending on the hardware. This first step needs to be done just once per each W90 file/material/direction combination. Subsequently, SC potential calculations and post-processing steps will load from *Hr+material+direction* folder created in this step.

Example 4.2: SC-potential energy calculation step with *BP-scp.py*

Once we have performed the pre-processing step, we can compute all the SC-potential calculations desired for the material(direction) system. In this particular example, it is STO(100). However, we should note that the chance of obtaining different results according to the initial DFT and Wannierization approach still remains. Before performing the calculation the SC-potential, we will just show the settings of the configuration file *scp.yaml*:

```yaml
# Config. file for the main component 'BP-scp.py'
#====================INPUT============================
---
SCP_CALCULATION:
        identifier : "sto01"
        material : "STO"
        crystal_face : "100"
        number_of_planes : 40
        shift_from_LUL : 0.008
        BC1_topmost_layer : -0.3
        BC2_in_bulk : 0.0
        Neumann_at_bulk : no
        sqrt_kgrid_numbers : 36
        k_shift :  [0.001, 0.001]
        temperature : 10
        mixing_factor : 0.09
        permittivity_model : "Cop"
        potential_live_visualization : yes
        error_live_visualization : no

        ADVANCED_PARAMETERS:
                conv_threshold : 1.0e-6
                max_iterations : 500
                Total_Hk_method : "vectorized"
                V_initial : "linear"
                cons_charge_background : no
                charge_per_site : 0.06
                charge_extension : 40
...
```

Following the example given in the main manuscript, we type:

$ python BP-scp.py -id run1 -mt STO -cfd 100 -tl 40 -nk 26 -bc1 -0.22

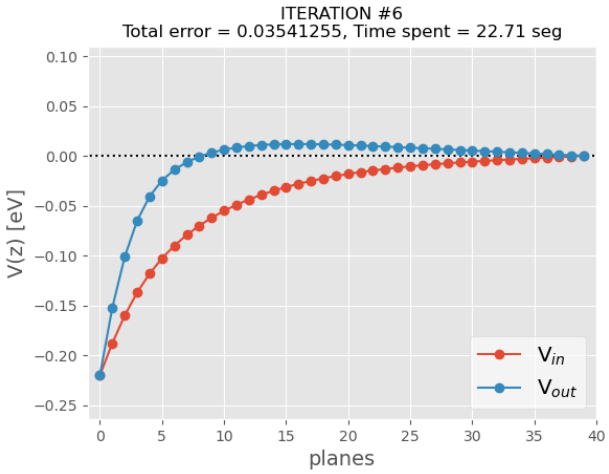
The parameters that we are modifying through the command-line are the identifier (*id*), the material (*mt*), the crystal face (*cfd*), the number of planes (*tl*), the square root of total k-grid points (*nk*) and the value of the potential energy at the top-most layer (*bc1*). Once the *BP-scp.py* component starts we will see immediately the details of the calculations. It will look like:

```
DETAILS:
        Identifier : run1
        Surface : STO(100)
        Number of planes : 40
        K-grid : 26 x 26
                K-shift : (0.001 ,0.001)
        Boundary conditions :
                V[0] = -0.22 eV
                V[L-1] = 0.0 eV
        Neumann condition at V[L-1] : False
        Permittivity model : Cop
        Temperature : 10 K
        Fermi level : 11.47650 eV
        Total Hk method : vectorized
        Mixing factor : 0.09
        Convergence threshold : 1e-06
        Using charge background : False
        Initial V shape : linear
```

Note that, as we mentioned at the beginning of this text, the input parameters are those introduced by command-line, whereas the omitted ones are those set in the configuration file.

If the live visualization option is set to ‘yes’, as indicated in the *scp.yaml* file above, we will observe the values of the potential energy during the iterative process, as shown in the figure below. We will have a stepwise visualization of the potential energy profiles along the

slab, before (V_{in}) and after (V_{out}) the Schrödinger-Poisson iteration. On the top of the plot, we can also see the number of iterations, the total error and the time spent in seconds.



After a successful calculation, we will have a folder named with the identifier, in this case the *run1* label. Inside the folder there will be three files: *run1_SCP.dat*, *run1.log* and *run1.yaml*. The *run1_SCP.dat* holds the SC solution of the problem and the columns have the following quantities: planes, SC-potential energy in eV, electron density in numbers of electrons /plane, electron density in numbers of electrons/cm^2, electric field in eV/m and relative permittivity. The *run1.log* file logs all the same information we get from the output in terminal. Finally, the *run1.yaml* file contains a Python dictionary holding the most relevant information of the SC calculation performed. It is worth inspecting this file:

```
BC1_topmost_layer: -0.22
BC2_in_bulk: 0.0
Fermi_level: 11.4765
crystal_face: '100'
identifier: run1
k_shift:
- 0.001
- 0.001
lattice_parameter: 3.9425
material: STO
number_of_planes: 40
sqrt_kgrid_numbers: 26
temperature: 10
```

As we can see, this file carries all the information needed to run any post-processing calculation by recalling the parameters from it.

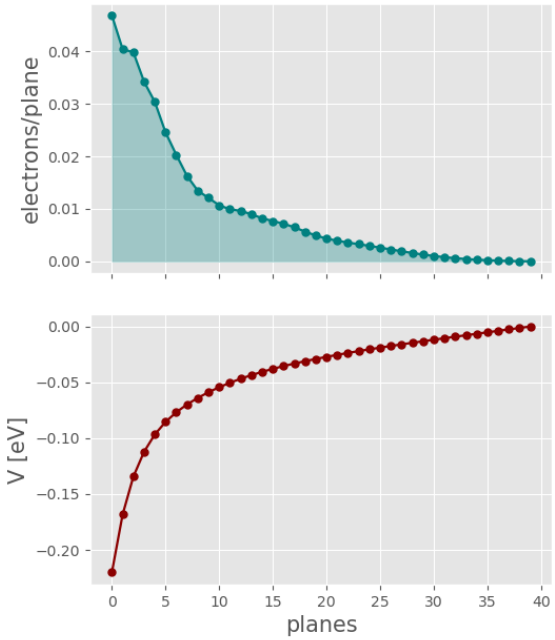
We can now repeat the calculation with the other boundary condition at the top-most layer (-0.36 eV), but this time with an updated identifier, otherwise an error will emerge.

$ python BP-scp.py -id run2 -mt STO -cfd 100 -tl 40 -nk 26 -bc1 -0.36

So far, we have computed two SC calculations with different boundary conditions at the top-most layer. The SC solutions can be quickly checked by using the *BP-fast_plot.py*. For the first SC solution we type:

$ python BP-fast_plot.py -id run1

and the following output plot will be obtained:



For the second one, we type:

$ python BP-fast_plot.py -id run2

and the following output plot will be obtained: