Example: comparison to ARPES data

This file is intended to detail the calculation presented in Sections 4.4 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

Stabilized 2DES on the bare (001) surface of STO

We will assume that the pre-processing step for STO(100) is already done. So, here it is how the *scp.yaml* configuration file looks like:

```yaml
---
SCP_CALCULATION:
        identifier : "sto1"
        material : "STO"
        crystal_face : "001"
        number_of_planes : 40
        shift_from_LUL : 0.008
        BC1_topmost_layer : -0.37
        BC2_in_bulk : 0.0
        Neumann_at_bulk : no
        sqrt_kgrid_numbers : 46
        k_shift :  [0.001, 0.001]
        temperature : 10
        mixing_factor : 0.09
        permittivity_model :  "1+4.5e3/(1+E/2.0e6)" # See below
        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
...
```

Now, we will compute a SC calculation under the “comp_exp” identifier (*id*), so:

$ python BP-scp.py -id comp_exp

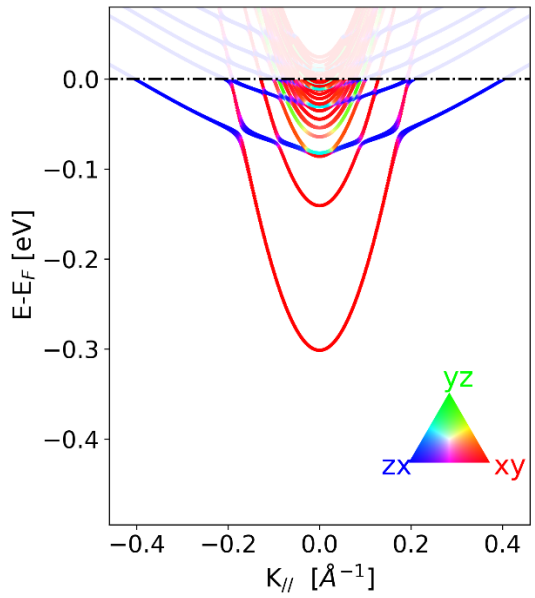
Note that, in this particular case, we are introducing a specific model for relative permittivity as input. Now, let’s see the *bands.yaml* file for the calculation of band structure projected onto the atomic orbitals (note that we are focusing on the main and orbital character block):

```
---
BAND_STRUCTURE :
        identifier : "sto01"
        path : "XGX"
        number_of_kpoints : 600
        reference_kpoint : "G"
        Total_Hk_method : 'vectorized'
        num_bands : 50
        bands_task : 0
        initial_plane : 0
        final_plane : 4
        skip_bands : 0

        TOTAL_BANDS :
        ORBITAL_CHARACTER :
                PLOT_ADJUST :
                        plotstyle : "default"
                        xy_limits : *limxy
                        color_seq :  r,lime,b
                        point_size : 2
                        fig_size : [6,6.9]
                        axis_adjust : [0.2, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                        title : ""
                        title_size : 18
                        shadow_above_Ef : 0.9
                        shadow_color : 'w'
                COLOR_TRIANGLE:
                        proportion : '35%'
                        location : 4
                        padding : 0.5
                        fontsize : 20
                LABELS :
                        xlabel : 'K$_{//}$  [$\AA^{-1}$]'
                        xfontsize : 18
                        ylabel : 'E-E$_{F}$ [eV]'
                        yfontsize : 18
                        ticksize : 17
                SAVING :
                        save_bands : no
                        save_plot : yes
                        format : '.png'
                        dpi : 300

        PLANE_PROJECTION :
...
```

We compute the desirable band structure as:

$ python BP-bands.py -id comp_exp -kp 1200 -tk 1 -xy -0.46 0.46 -0.495 0.08

Given that we set a value different than 0.0 in the “shadow_above_Ef” keyword, we obtain translucent bands above the Fermi level. The output plot we get is:



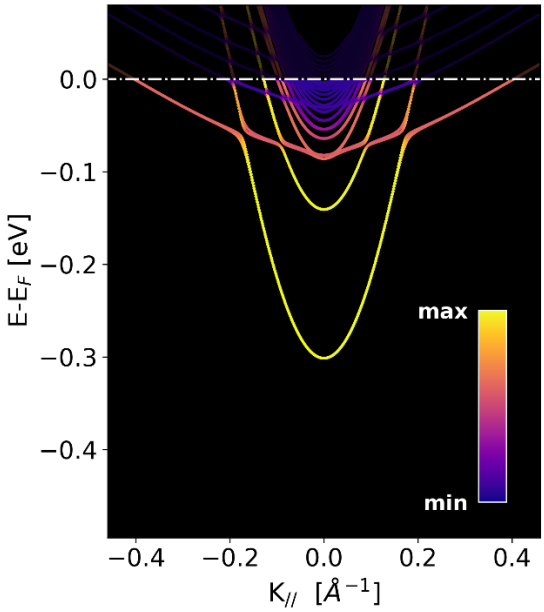
To compute the projection onto the first four planes we first unwrap and take a look at the PLANE_PROJECTION block in *bands.yaml*.

```
16          TOTAL_BANDS :
40          ORBITAL_CHARACTER :
69          PLANE_PROJECTION :
70                  PLOT_ADJUST :
71                          plotstyle : "default"
72                          xy_limits : *limxy
73                          colormap : "plasma"
74                          background_color : "k"
75                          point_size : 1.0
76                          fig_size : [6.0,6.9]
77                          axis_adjust : [0.2, 0.15, 0.9, 0.9] #[left, bottom, right, top]
78                          title : ""
79                          title_size : 18
80                          shadow_above_Ef : 0.7
81                          shadow_color : 'k'
82                  COLORBAR :
83                          location : [0.8, 0.2, 0.045, 0.27] #[x, y, width, height]
84                          textbar : ['min','max']
85                          fontsize : 18
86                          fontcolor : 'w'
87                  LABELS :
88                          xlabel : 'K$_{//}$  [$\AA^{-1}$]'
89                          xfontsize : 18
90                          ylabel : 'E-E$_{F}$ [eV]'
91                          yfontsize : 18
92                          ticksize : 17
93                  SAVING :
94                          save_bands : no
95                          save_plot : yes
96                          format : '.png'
97                          dpi : 300
98  ...
```

Then, we can proceed by typing:

$ python BP-bands.py -id comp_exp -kp 1200 -tk 2 -xy -0.46 0.46 -0.495 0.08

Note that, in this case, we omit to enter the values for initial and final planes, because they are already set to the values we want in the main block of *bands.yaml* file. The corresponding output plot will be:



Fermi surface calculation

The *energy_slices.yaml* configuration file looks:

```
---
ENERGY_SLICES :
        identifier : "sto_01"
        sqrt_kgrid_numbers : 400
        kbox_factor : 0.6
        kbox_shift : [0.0,0.0]
        win_energy_calc : 0.04
        batches : 100
        energy_cut : 0.0
        outfile : "default"
...
```

We will run the component *energy_slices.py* to get the Fermi surface just as follows:

$ python BP-energy_slices.py -id comp_exp

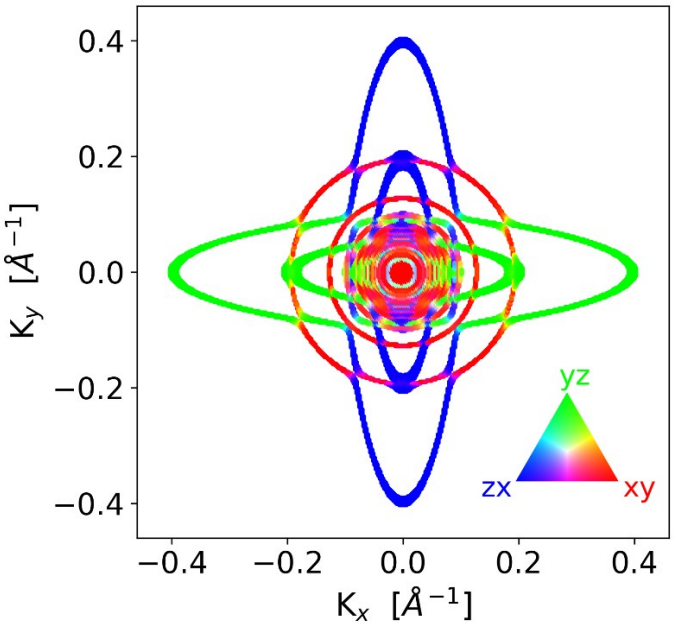
When the run finishes you can plot the Fermi surface using the *BP-energy_plot.py*. Firstly, we take a look at the *energy_plot.yaml* file:

```
---
ENERGY_PLOTTER :
        identifier : "sto_01"
        input_file : "default"
        orbital_char : yes
        color_seq : red,lime,blue
        energy_window : 0.004

        PLOT_ADJUST :
                plotstyle : 'default'
                point_size : 3
                xy_limits : [-0.46, 0.46, -0.46, 0.46] #[x_min, x_max, y_min, y_max]
                fig_size : [6,6]
                axis_adjust : [0.2, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                title : ""
                title_size : 12
        LABELS :
                xlabel : 'K$_x$  [$\AA^{-1}$]'
                xfontsize : 18
                ylabel : 'K$_y$  [$\AA^{-1}$]'
                yfontsize : 18
                ticksize : 17
        COLOR_TRIANGLE:
                proportion : '35%'
                location : 4
                padding : 0.5
                fontsize : 16
        SAVING :
                save_plot : yes
                format : ".png"
                dpi : 300
```

Finally, we can obtain the Fermi surface plot by typing:

$ python BP-energy_plot.py -id comp_exp

The output plot we get is:



Giant Rashba-type spin-splitting in the bismuth tellurohalide BiTeBr

Firstly, we are going to indicate the pre-processing step. We included a W90 file to do calculations with BiTeBr, it is named *BTB_hr.dat*. So, by typing:

$ python BP-preproc.py -mt BTB -cfd 001

It will generate the folder *HrBTB001h* with the plane separation. The character '*h*' at the end of the confinement direction is to emphasize that now we are dealing with a hexagonal lattice.

Before performing the SC calculation, we take a look at the settings in the configuration file *scp.yaml*:

```yaml
---
SCP_CALCULATION:
        identifier : "btb1"
        material : "BTB"
        crystal_face : "001"
        number_of_planes : 30
        shift_from_LUL : 0.027
        BC1_topmost_layer : -0.58
        BC2_in_bulk : 0.075
        Neumann_at_bulk : no
        sqrt_kgrid_numbers : 46
        k_shift :  [0.001, 0.001]
        temperature : 10
        mixing_factor : 0.07
        permittivity_model :  "cte_12" # See below
        potential_live_visualization : yes
        error_live_visualization : no

        ADVANCED_PARAMETERS:
                conv_threshold : 1.0e-5
                max_iterations : 500
                Total_Hk_method : "vectorized"
                V_initial : "linear"
                cons_charge_background : no
                charge_per_site : 0.06
                charge_extension : 40
```

We already set all the necessary parameters in the configuration file. It is worth mentioning that, in order to save time, we inferred the *bc2* parameter for Dirichlet B.C. by trial and error, using the Neumann B.C. without reaching convergence. Said this, to perform the SC calculation just type:

$ python BP-scp.py

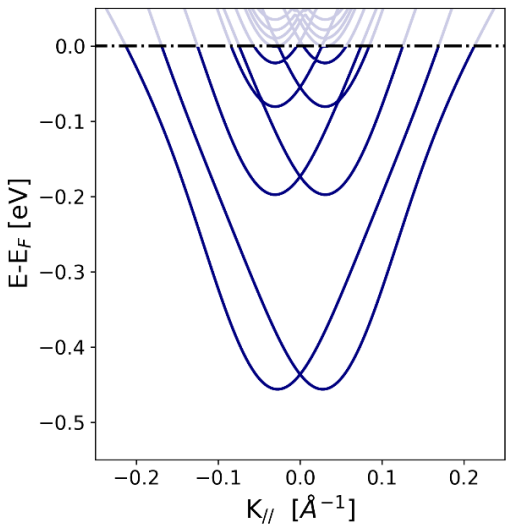
Once the SC solution is converged, we can step into the post-processing. For the band structure, again we take a look at the configuration file *bands.yaml*:

```
---
BAND_STRUCTURE :
        identifier : "btb1"
        path : "KGK"
        number_of_kpoints : 600
        reference_kpoint : "G"
        Total_Hk_method : 'vectorized'
        num_bands : 50
        bands_task : 0
        initial_plane : 0
        final_plane : 4
        skip_bands : 0

        TOTAL_BANDS :
                PLOT_ADJUST :
                        plotstyle : "default"
                        xy_limits : &limxy [-0.25, 0.25, -0.55, 0.05]  #[x_m
                        linecolor : "navy"
                        linewidth : 2.0
                        fig_size : [6,6]
                        axis_adjust : [0.22, 0.15, 0.9, 0.9] #[left, bottom, r
                        title : ""
                        title_size : 18
                        shadow_above_Ef : 0.8
                        shadow_color : 'w'
                LABELS :
                        xlabel : 'K$_{//}$  [$\AA^{-1}$]'
                        xfontsize : 18
                        ylabel : 'E-E$_{F}$ [eV]'
                        yfontsize : 18
                        ticksize : 14
                SAVING :
                        save_bands : no
                        save_plot : yes
                        format : '.png'
                        dpi : 300

        ORBITAL_CHARACTER :
        PLANE_PROJECTION :
```

The total band structure can be computed by typing:

$ python BP-bands.py

And we obtain the output plot:



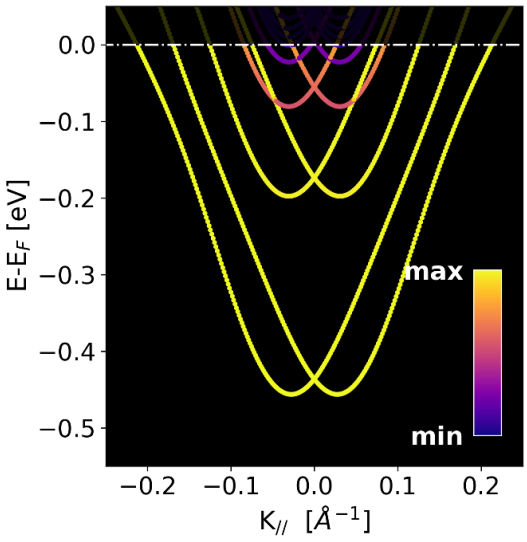
To compute the projection onto the first four plane, first we inspect the PLANE PROJECTION block in *bands.yaml*.

```
16  TOTAL_BANDS :
40  ORBITAL_CHARACTER :
69  PLANE_PROJECTION :
70      PLOT_ADJUST :
71          plotstyle : "default"
72          xy_limits : *limxy
73          colormap : "plasma"
74          background_color : "k"
75          point_size : 6
76          fig_size : [6.0,6.9]
77          axis_adjust : [0.2, 0.15, 0.9, 0.9] #[left, bottom, right, top]
78          title : ""
79          title_size : 18
80          shadow_above_Ef : 0.8
81          shadow_color : 'k'
82      COLORBAR :
83          location : [0.82, 0.2, 0.045, 0.27] #[x, y, width, height]
84          textbar : ['min','max']
85          fontsize : 18
86          fontcolor : 'w'
87      LABELS :
88          xlabel : 'K$_{//}$  [$\AA^{-1}$]'
89          xfontsize : 18
90          ylabel : 'E-E$_{F}$ [eV]'
91          yfontsize : 18
92          ticksize : 17
93      SAVING :
94          save_bands : no
95          save_plot : yes
96          format : '.png'
97          dpi : 300
98  ...
```

Note that, in the main block above we already have by default the projection onto the first four planes. In consequence, all we need to do to get the projected bands is to type:

$ python BP-bands.py -tk 2 -kp 1200

where, apart from changing the task to perform, we increased the k-points to improve the resolution. The output plot is:



Finally, to compute the Fermi surface we first take a look at the *energy_slices.yaml* and *energy_plot.yaml* configuration files.

```
# Input file for 'BP-energy_slices.py' component
#==================INPUT=========================
---
ENERGY_SLICES :
                identifier : "btb1"
                sqrt_kgrid_numbers : 700
                kbox_factor : 0.5
                kbox_shift : [0.0,0.0]
                win_energy_calc : 0.04
                batches : 100
                energy_cut : 0.0
                outfile : "default"
...
```

```
# Input file for 'BP-energy_plot.py' component
#==================INPUT=========================
---
ENERGY_PLOTTER :
        identifier : "btb1"
        input_file : "default"
        orbital_char : no
        color_seq : navy,lime,blue
        energy_window : 0.006

        PLOT_ADJUST :
                plotstyle : 'default'
                point_size : 1
                xy_limits : [-0.25, 0.25, -0.25, 0.25] #[x_min, x_max, y_min, y_max]
                fig_size : [6,6]
                axis_adjust : [0.2, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                title : ""
                title_size : 12
        LABELS :
                xlabel : 'K$_x$  [$\AA^{-1}$]'
                xfontsize : 18
                ylabel : 'K$_y$  [$\AA^{-1}$]'
                yfontsize : 18
                ticksize : 17
        COLOR_TRIANGLE:
                proportion : '35%'
                location : 4
                padding : 0.5
                fontsize : 16
        SAVING :
                save_plot : yes
                format : ".png"
                dpi : 300
...
```

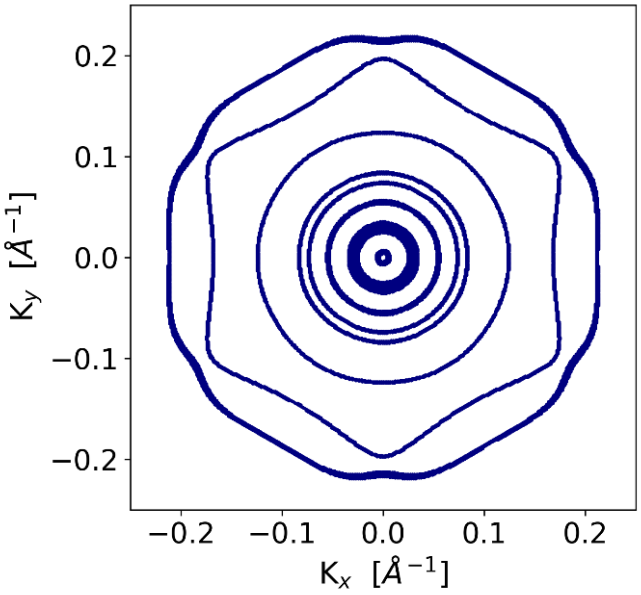
So, with the parameters already set, to get the file with the Fermi surface data we just type:

$ python BP-energy_slices.py

Note, from the *energy_slices.yaml* file, that we are using a relative dense k-grid of 700 x 700 points, so that this calculation will take a while. If these settings overload your hardware, the computational cost can be reduced by increasing the batches. Once this calculation finishes, you can plot the Fermi surface by typing:

$ python BP-energy_plot.py

and the outplot plot will be:


0.2
0.1
0.0
−0.1
−0.2
K_y [$Å^{-1}$]
−0.2
−0.1
0.0
0.1
0.2
K_x [$Å^{-1}$]