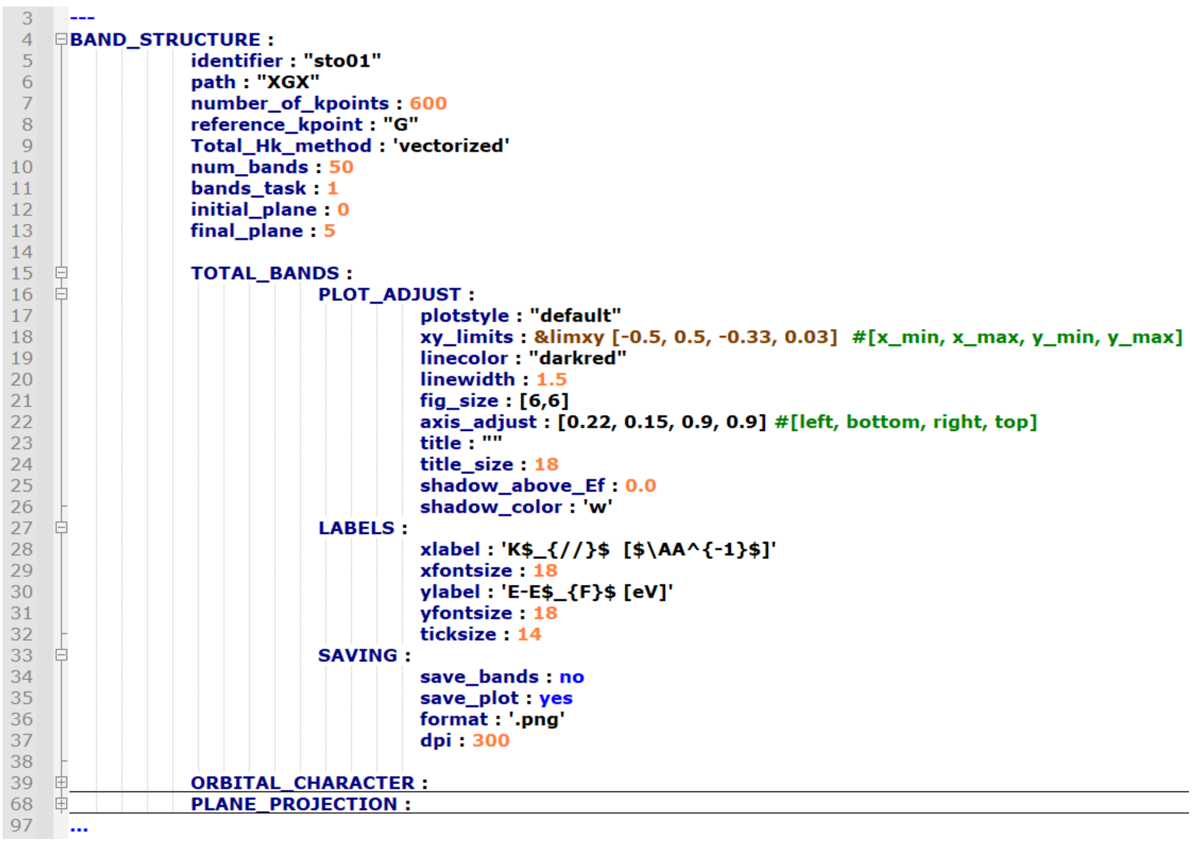
Example: band structure calculations

This file is intended to detail the calculation presented in Sections 4.3.1 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

Total bandstructure calculation

First of all, we show the *bands.yaml* file. In order to make it more readable, we show at first the main block along with the total bandstructure calculation block.

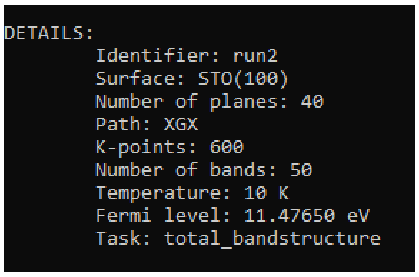


```
---
BAND_STRUCTURE :
            identifier : "sto01"
            path : "XGX"
            number_of_kpoints : 600
            reference_kpoint : "G"
            Total_Hk_method : 'vectorized'
            num_bands : 50
            bands_task : 1
            initial_plane : 0
            final_plane : 5

            TOTAL_BANDS :
                         PLOT_ADJUST :
                                   plotstyle : "default"
                                   xy_limits : &limxy [-0.5, 0.5, -0.33, 0.03]  #[x_min, x_max, y_min, y_max]
                                   linecolor : "darkred"
                                   linewidth : 1.5
                                   fig_size : [6,6]
                                   axis_adjust : [0.22, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                                   title : ""
                                   title_size : 18
                                   shadow_above_Ef : 0.0
                                   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 :
...
```

We will compute the band structure along a high symmetry path by typing:

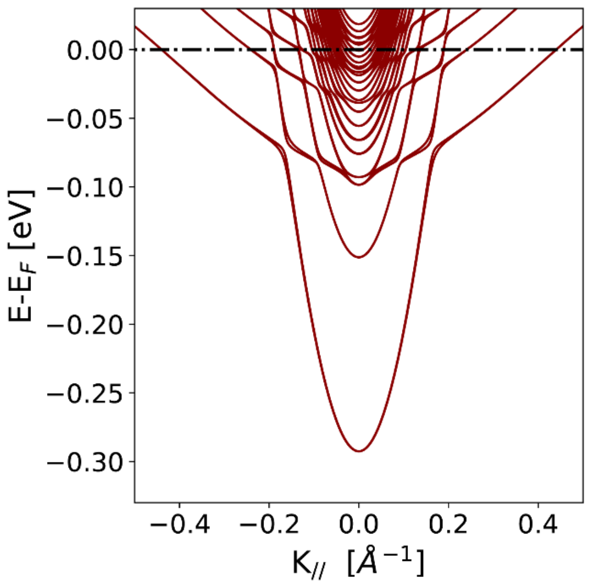
$ python BP-bands.py -id run2 -ph XGX -kp 600 -tk 0 -nb 50

The first parameter, *id*, is the identifier for the calculation that is being recalled. Then we have the path in reciprocal space (*ph*), the total number of points along the path (*kp*), the task to perform (*tk*) and the number of bands to compute (*nb*). In this case, the output in terminal looks like:



```
DETAILS:
        Identifier: run2
        Surface: STO(100)
        Number of planes: 40
        Path: XGX
        K-points: 600
        Number of bands: 50
        Temperature: 10 K
        Fermi level: 11.47650 eV
        Task: total_bandstructure
```

Note that, as we indicated 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. It is worth mentioning that the data belonging to the SC calculation is automatically loaded from the *run2.yaml* file within the *run2* folder. As we can see, setting the task to 0 printed it as "total_bandstructure", this is just one of the possible modes in which the bandstructure could be computed. At the end of this calculation, the total bandstructure along the selected path will be interactively shown.

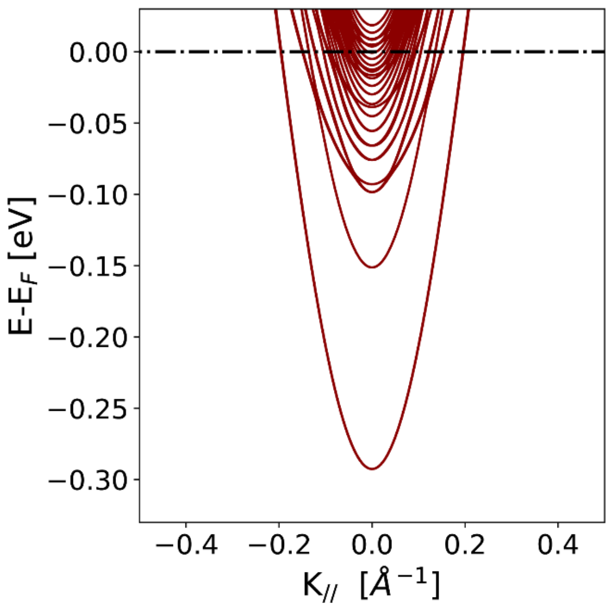


If the parameters "save_bands" and/or "save_plot" are setting to "yes", so the output .dat file with the bands and/or the plot will be saved to the *run2* folder.

We will now compute the bandstructure along another path:

$ python BP-bands.py -id run2 -ph MGM -kp 600 -tk 0 -nb 50

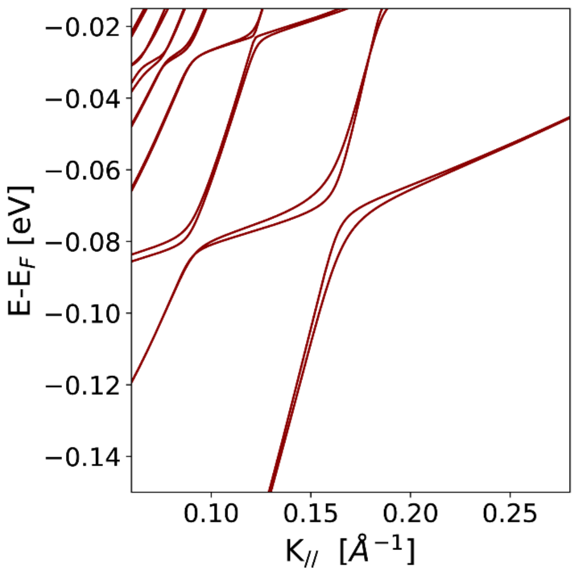
We will get the interactive plot showing the new computed bands.



We will now take a look at one of the avoided crossings along the Γ-X path, where the unconventional Rashba spin splitting appears. So, we type:

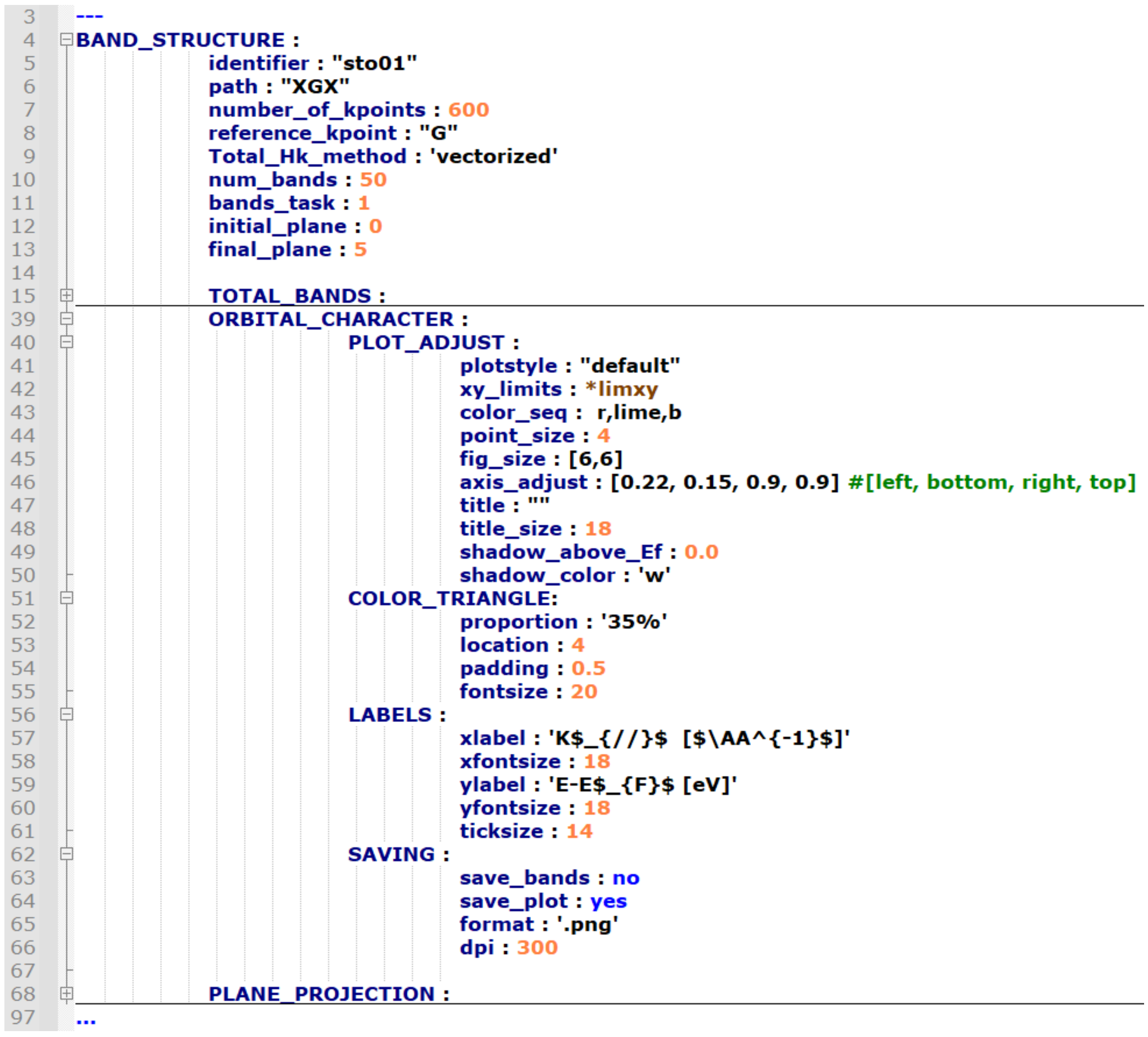
$ python BP-bands.py -id run2 -ph GX -kp 600 -tk 0 -nb 50 -xy 0.06 0.28 -0.15 -0.015

Note that we increased the resolution by choosing a shorter path while keeping constant the number of k-points. Here, the xy parameter was used, which sets the x and y values of the energy-momentum plot window. The output plot is now:



Bandstructure projection onto the manifold orbitals

This is how the part of the *bands.yaml* file for this projection looks like:

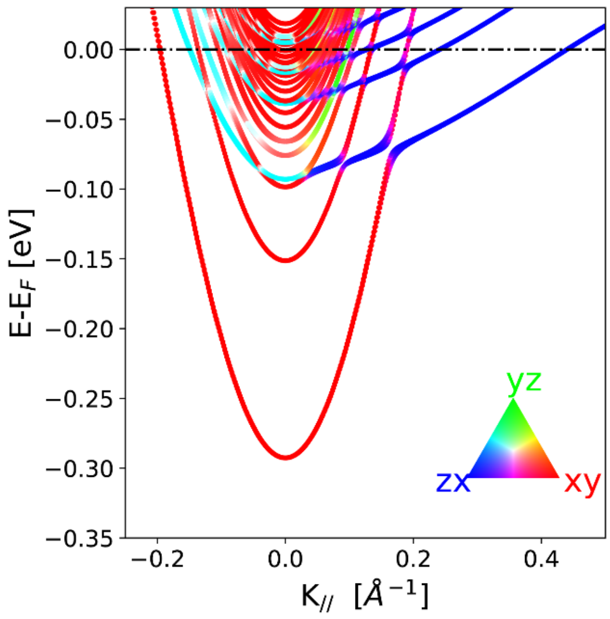


```
---
BAND_STRUCTURE :
        identifier : "sto01"
        path : "XGX"
        number_of_kpoints : 600
        reference_kpoint : "G"
        Total_Hk_method : 'vectorized'
        num_bands : 50
        bands_task : 1
        initial_plane : 0
        final_plane : 5

        TOTAL_BANDS :
        ORBITAL_CHARACTER :
                PLOT_ADJUST :
                        plotstyle : "default"
                        xy_limits : *limxy
                        color_seq :  r,lime,b
                        point_size : 4
                        fig_size : [6,6]
                        axis_adjust : [0.22, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                        title : ""
                        title_size : 18
                        shadow_above_Ef : 0.0
                        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 : 14
                SAVING :
                        save_bands : no
                        save_plot : yes
                        format : '.png'
                        dpi : 300

        PLANE_PROJECTION :
...
```

We are going now to compute the bands with projections onto the different orbitals in the *t2g* manifold by typing:

$ python BP-bands.py -id run2 -ph MGX -kp 1000 -tk 1 -nb 50 -xy -0.25 0.5 -0.35 0.03

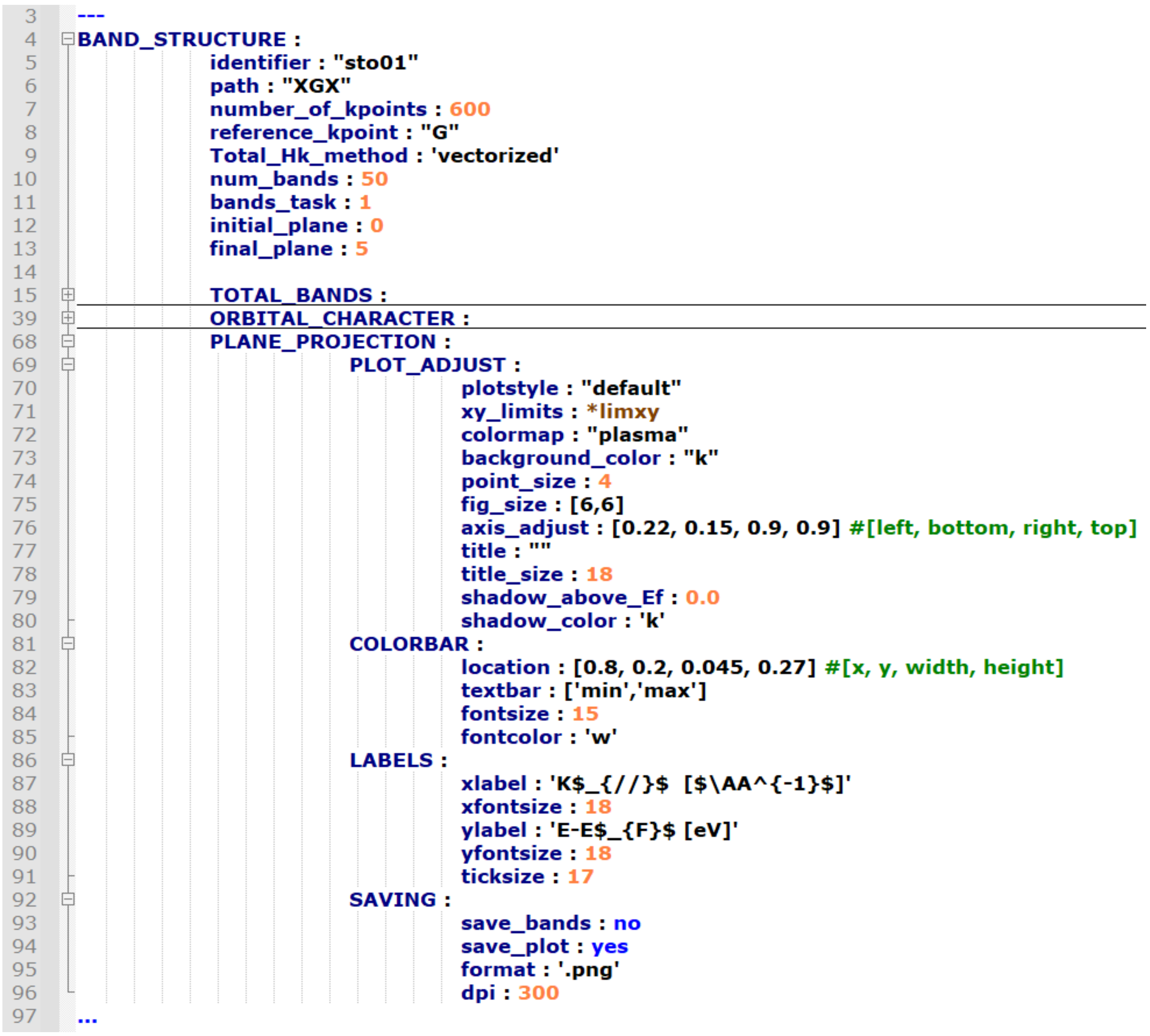
In this case, we are selecting the path MGX with a denser number of points. The task parameter equal to 1 indicates that we are projecting the bands onto the different orbitals. The output plot is:



0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
E-E$_F$ [eV]
−0.2
0.0
0.2
0.4
K$_{//}$ [$\AA^{-1}$]
yz
zx
xy

Bandstructure projection onto a set of planes

This is how the part of the *bands.yaml* file for this projection looks like:

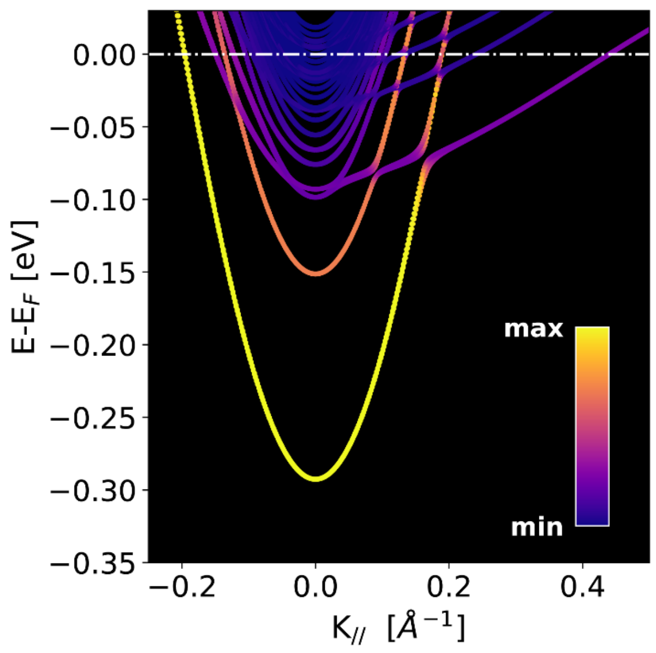


```
---
BAND_STRUCTURE :
            identifier : "sto01"
            path : "XGX"
            number_of_kpoints : 600
            reference_kpoint : "G"
            Total_Hk_method : 'vectorized'
            num_bands : 50
            bands_task : 1
            initial_plane : 0
            final_plane : 5

            TOTAL_BANDS :
            ORBITAL_CHARACTER :
            PLANE_PROJECTION :
                        PLOT_ADJUST :
                                  plotstyle : "default"
                                  xy_limits : *limxy
                                  colormap : "plasma"
                                  background_color : "k"
                                  point_size : 4
                                  fig_size : [6,6]
                                  axis_adjust : [0.22, 0.15, 0.9, 0.9] #[left, bottom, right, top]
                                  title : ""
                                  title_size : 18
                                  shadow_above_Ef : 0.0
                                  shadow_color : 'k'
                        COLORBAR :
                                  location : [0.8, 0.2, 0.045, 0.27] #[x, y, width, height]
                                  textbar : ['min','max']
                                  fontsize : 15
                                  fontcolor : 'w'
                        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
...
```

In this last example, we will project the bandstructure onto a set of planes. Consider the following line:

$ python BP-bands.py -id run2 -ph MGX -kp 1000 -tk 2 -nb 50 -xy -0.25 0.5 -0.35 0.03 -pi 0 -pf 2

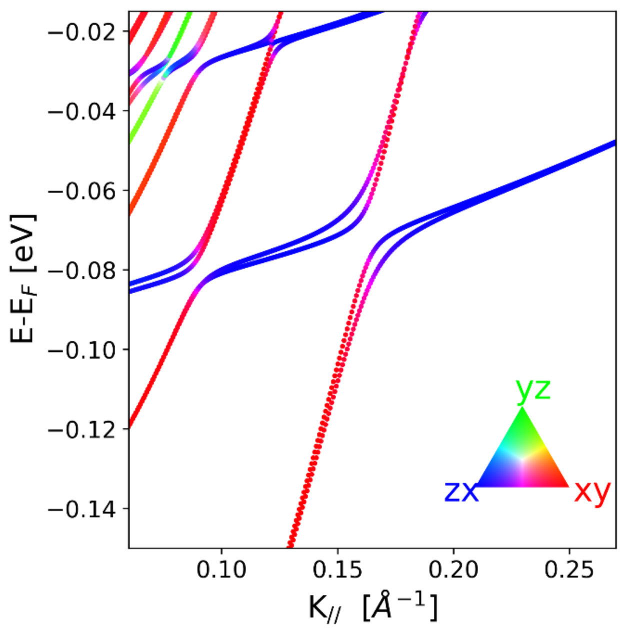
As we can note, the line is almost identical to the last example except for the task parameter, which is equal to 2 for the projection onto planes, and the two additional parameters: initial plane (*pi*) and final plane (*pf*). These new parameters allow for projecting the bandstructure onto the set given by [*pi*, *pf*-1] planes. In consequence, the projection will be done onto the two first planes. The output plot is:



To get a close-up view in the avoided crossing region, the user can type:

```
$ python BP-bands.py -id run2 -ph GX -kp 1200 -tk 1 -nb 50 -xy 0.06 0.27 -0.15 -0.015
```

Note that now we just specify the G-X path. Additionally, the discretization along the path was increased with *kp* equal to 1200 and the list of values of the *xy* parameter is enclosing the region of interest. The output plot is:



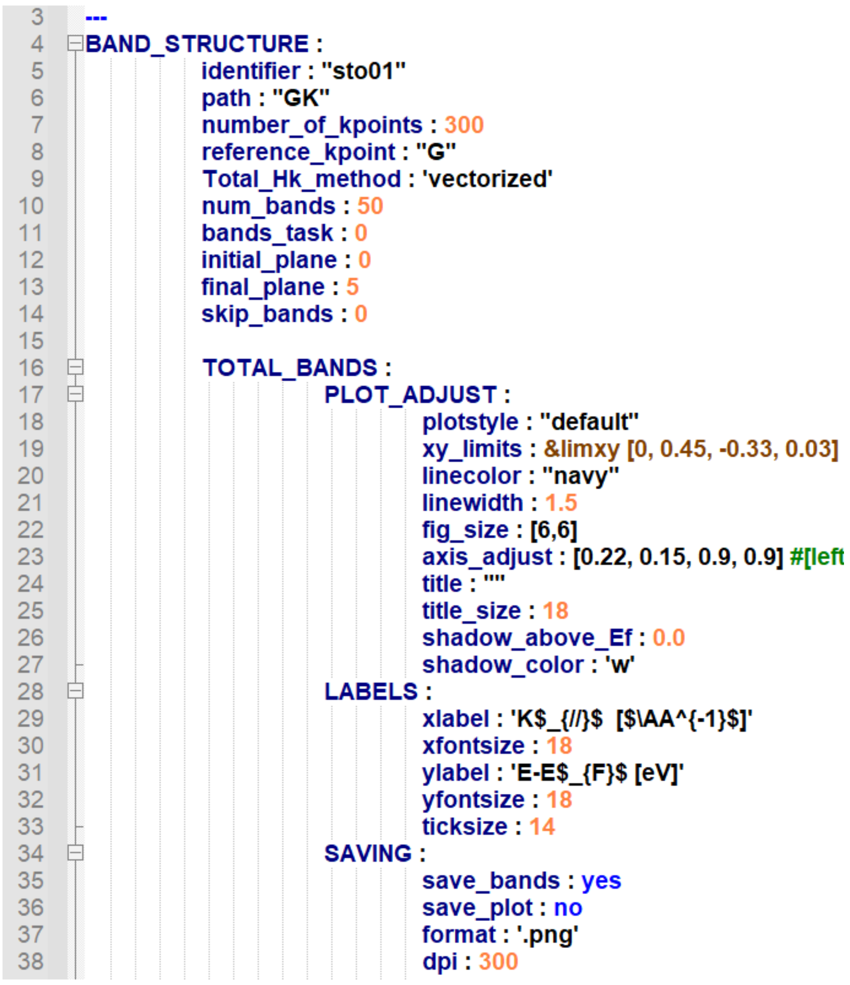
Comparison between STO with Ti t_{2g} manifold and STO with Ti t_{2g} + O 2p manifolds

The W90 file containing the STO with the Wannierization of the Ti t_{2g} and the O *2p* manifold is what we called *STOB_hr.dat*, being *STOB* the name adopted for this system in the dictionary within *BPdatabase.py* module. The calculations with STOB will be quite expensive, because of the 24-elements MLWF basis (18 O *2p* + 6 t_{2g} orbitals), so that, in order to reduce the computational load, we must use the method "iterable" in the *scp.yaml* as well as *bands.yaml*. Therefore, as shown in examples 4.1 and 4.2, the analogous lines for the pre-processing and the SC-calculation will be:

```
$ python BP-preproc.py -mt STOB -cfd 001
```

```
$ python BP-scp.py -id run2B -mt STOB -cfd 001 -tl 40 -nk 26 -bc1 -0.36
```

Note that now we are using the material *STOB* and we named the identifier as *run2B*, to compare it to the previous calculation under the identifier *run2* (Example 4.2). To compare the band structures between the *run2* (Ti t_{2g}) and the *run2B* (Ti t_{2g} + O *2p*) we start with the following settings in the configuration file *bands.yaml*:

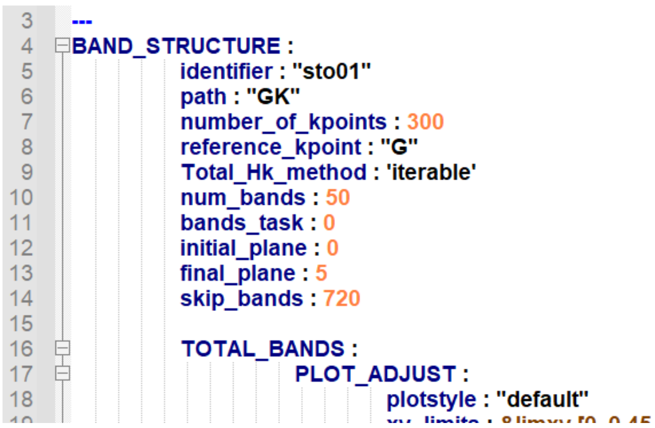


```
---
BAND_STRUCTURE :
          identifier : "sto01"
          path : "GK"
          number_of_kpoints : 300
          reference_kpoint : "G"
          Total_Hk_method : 'vectorized'
          num_bands : 50
          bands_task : 0
          initial_plane : 0
          final_plane : 5
          skip_bands : 0

          TOTAL_BANDS :
                    PLOT_ADJUST :
                            plotstyle : "default"
                            xy_limits : &limxy [0, 0.45, -0.33, 0.03]
                            linecolor : "navy"
                            linewidth : 1.5
                            fig_size : [6,6]
                            axis_adjust : [0.22, 0.15, 0.9, 0.9] #[left
                            title : ""
                            title_size : 18
                            shadow_above_Ef : 0.0
                            shadow_color : 'w'
                    LABELS :
                            xlabel : 'K$_{//}$  [$\AA^{-1}$]'
                            xfontsize : 18
                            ylabel : 'E-E$_{F}$ [eV]'
                            yfontsize : 18
                            ticksize : 14
                    SAVING :
                            save_bands : yes
                            save_plot : no
                            format : '.png'
                            dpi : 300
```

Then we run:

```
$ python BP-bands.py -id run2
```

Note that we are computing the band structure just along Γ-X path with 300 k-points. We are also saving the *.dat* file of the bands. In the new version of *BinPo* (v1.1) we add in this configuration file the parameter *skip_bands* that can be used to skip a number of lower energy bands in the plotting. To repeat this calculation for the run2B case, we just modified the main block of the file as follows:



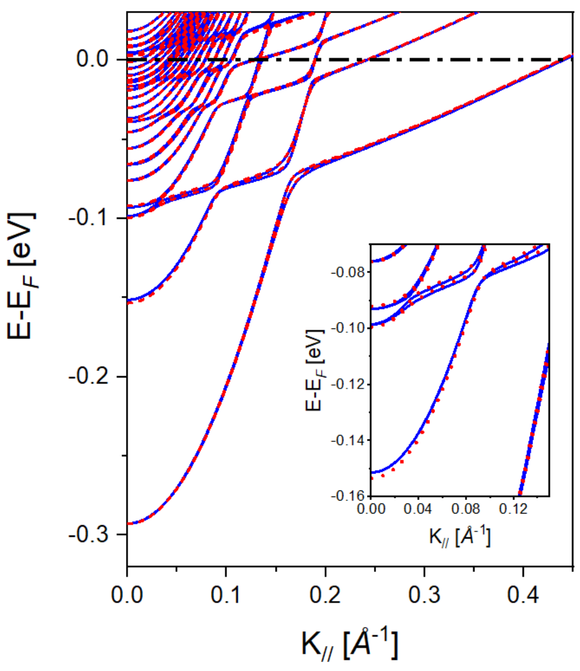
```
---
BAND_STRUCTURE :
            identifier : "sto01"
            path : "GK"
            number_of_kpoints : 300
            reference_kpoint : "G"
            Total_Hk_method : 'iterable'
            num_bands : 50
            bands_task : 0
            initial_plane : 0
            final_plane : 5
            skip_bands : 720

            TOTAL_BANDS :
                        PLOT_ADJUST :
                                    plotstyle : "default"
```

As can be seen, we changed the method to "iterable" and set *skip_bands* = 720 (18 MLWFs from oxygen x 30 planes) to skip the entire O *2p* manifold, because it is not necessary in this comparison. We compute the band structure for *run2B* as:

$ python BP-bands.py -id run2B

It could take a few minutes, depending on your hardware.

Finally, we have to *.dat* files with the band structure for both cases to generate a plot like the one we had shown in Fig. 7 of the main text that we reproduce here below.


E-E$_F$ [eV]
K$_{//}$ [Å^{-1}]
E-E$_F$ [eV]
K$_{//}$ [Å^{-1}]