Wannier functions — band disentangling

Klaus Koepernik

July 4, 2022

Contents

1 Introduction

We assume that the user already got acquainted with the FPLO Wannier function (WF) module, so that we do not have to explain the basics. Here we focus on the aspect of band disentangling, which plays an important role for `pyfplo`. The `slabify` module of `pyfplo` uses the file `+hamdata` as input, which in turn is produced by the WF module. In most applications the relevant bands are not neatly separated by gaps.

We will walk you through an iteration of intermediate results, which demonstrates how to approach the problem in a semi systematic way.

The solution will be found in the `solution` directory. You will work in the `task` directory.

2 Bcc Fe: 3d-only Wannier functions

In the beginning we will produce 3d-only Wannier functions for bcc-Fe. We will use a full-relativistic calculation to demonstrate (as a side effect of the tutorial) how this is done. So, please go into the task directory.

- Create a directory and copy files:

```
mkdir bccFe
cd bccFe
cp ../=.in .
cp ../=.dens .
```

- Open `fedit` and switch on `bandplot ▷ bandstructure plot` and `bandplot ▷ weights`. Go into the main menu and switch on `use data directories` and `Quit/Save`.
- Run

  ```
  fplo22.00-62-x86_64 | tee out
  faddwei22.00-62-x86_64
  ```

- Edit `=.addwei` to look like this

  ```
  weightinfile +bweightslms
  weightoutfile +bwsum


  name Fe3dup
      atom Fe sites 1 orbitals 3d spin up

  name Fe3ddn
      atom Fe sites 1 orbitals 3d spin dn

  name Fe4s
      atom Fe sites 1 orbitals 4s spin both

  name Fe4p
      atom Fe sites 1 orbitals 4p spin both
  ```

  Pay attention to `weightinfile` and `spin`.
- Run

  ```
  faddwei22.00-62-x86_64
  ```

  again and open

  ```
  xfbp +bwsum
  ```

- Open `Edit ▷ Sets` go to the `Weights` tab and check all weights followed by `Apply` to make them all visible:

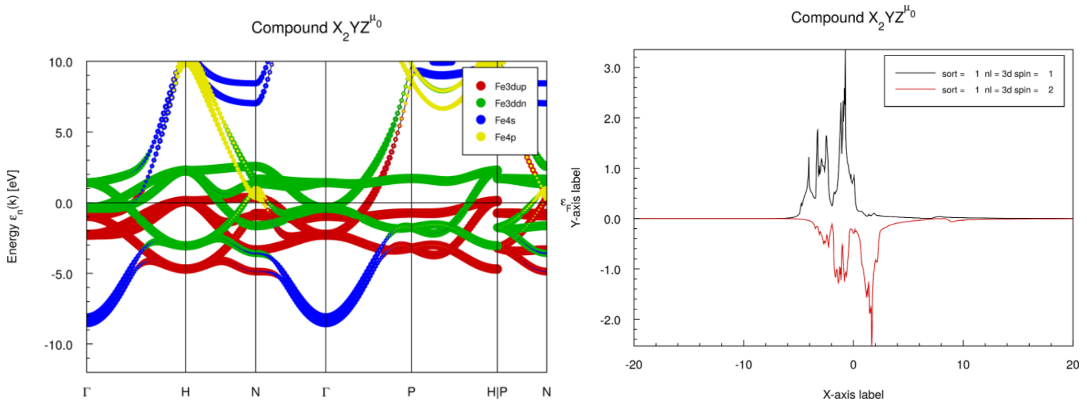

Figure 1: The fatbands and 3d-dos of bcc Fe.

- Close and open

  ```
  xfbp +dos/+dos.sort001.nl005
  ```

  See Fig. 1. You will notice a little bump around 8–9eV in the DOS and clearly some 3d-weight in the free-electron like band along ΓP (2–15eV). This is a 3d-4s/4p hybridization tail.

The little exercise above should always be done before creating a WF fit. Afterall, we need to know where our bands are. Please notice the extend of the 3d bands (roughly $[-5, 3]$eV).

We will **not** create `=.wandef` by hand. The reasons are simple:

- We want to write as little as possible.
- We want to avoid missing orbitals.
- For slabify we need always whole sets of orbitals, mostly a complete set of a minimum basis. Such a basis is defined as spd for d-metals and sp for main group elements.
- Note, that for topological properties it is advised to include all orbitals of the topological band complex including bands above the topological bands, since especially for topological insulators it it possible to hide the topological properties by a wrong choice of WFs (topological obstruction) — use a minimum basis!
- Finally, the choice of WFs shall not destroy the symmetry which protects the topological properties. E.g. for TIs one can find a minimal set of WFs which describe the few relevant bands but break time reversal symmetry. These can be totally valid WFs, but the topology, derived from such a set of WFs can be different — use a minimum basis.
- An exception of the points above would be a situation where e.g. Bi-6p bands form an isolated band complex around the Fermi level. If these bands have non-zero p-character at each k-point in the BZ it is possible to use a p-only WF set for topological properties.

- Copy a file

  ```
  cp ../makewandef3d.py .
  ```

- Edit it to look like this

Listing 1: makewandef3d.py

```
1  #! /usr/bin/env python
2
3  import pyfplo.wanniertools as wt
4
5
6  # ===================================================================
7
8  if __name__ == '__main__':
9
10
11     wdc=wt.WanDefCreator(rcutoff=25,wftol=0.001,coeffformat='bin',
12                          wfgriddirections=[[2,0,0],[0,2,0],[0,0,2]],
13                          wfgridsubdiv=[1,1,1],savespininfo=True)
14
15     emin=-5
16     emax= 3
17     delower=1
18     deupper=1
19
20     wdc.add(wt.MultipleOrbitalWandef('Fe',[1],['3db'],
21                                      emin=emin,emax=emax,
22                                      delower=delower,deupper=deupper))
23
24     wdc.writeFile('=.wandef')
```

Note lines 15-18 and 20 (`3db` means all 3d orbitals in real harmonics both spins). The sites list `[1]` contains only one site, but it could contain more sites, if there were more Fe sites in the compound.

Also consult `FPLO22.00-62/DOC/pyfplo/pyfplo.pdf`.

- Now, run the script. (If a syntax error occurs, due to editing mistakes, correct them until you can execute the script without errors.)

  ```
  python makewandef3d.py
  ```

  Have a look at the created `=.wandef`.

- Run fplo

  ```
  fplo22.00-62-x86_64 | tee out
  ```

  We should have `+wancoeff` now.

- Re-run fplo

  ```
  fplo22.00-62-x86_64 | tee outwf
  ```

- and when done, interrupt it with `Ctrl-C`.

  It created `+wancoeffbin`. (This file is smaller than `+wancoeff` and loads faster. One could in principle delete `+wancoeff` now but be aware that the binary file might not be working on another machine. In other words `+wancoeffbin` might not be transferable.).

  Furthermore, we have some new subdirectories and the usual files `+wanband`, `+wanbandtb`, `+wanbweights`, `WF.net` and so forth.

- There is a generally very usefull little script. Let's copy the file:

```
cp ../wband.xpy .
```

- and execute

```
xfbp wband.xpy
```

  to obtain the left panel of Figure 2.

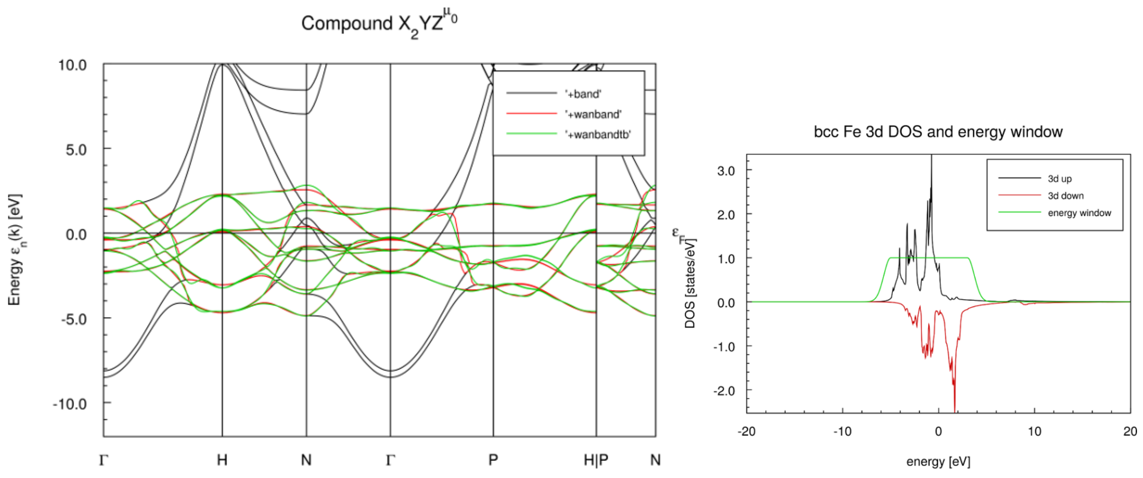

Figure 2: band, wanband and wanbandtb. The right panel shows the energy window for orientation.

Note that the highest WF band must connect downwards on the way from Γ to H. This is so since we cannot describe the free-electron like band, which crosses the 3d-bands with a 3d-only basis. Also note the wiggliness. This is owed to the fact that our energy window is narrow and that the free-electron band contains 3d-weight and most importantly that a narrow window means that when going through the free-electron band region the highest band leaves the window while the lowest enters the window. These are rather abrupt changes in the Hilbert space underlying the WF projectors.

We try to cure this by doing something quite unexpected. We increase the window to contain the free-electron bands.

But before we do this let's introduce some work flow aspects. We will repeatedly edit `makewandef3d.py`, execute fplo and look at the bands. So, let's do this more efficiently.

- Open another terminal, change into the bccFe directory and execute

```
fplo22.00-62-x86_64 | tee outwf
```

  Do NOT hit `Ctrl-C` as the program says. Leave it alone.

- In the original window execute (once and for all)

```
xfbp wband.xpy &    # the & is important
```

In the original terminal you can now edit/save and execute `makewandef3d.py`.
In the fplo-terminal you can hit `enter` to re-run the WF calculation and in xfbp you can hit `Ctrl-R` to reload the files.

Ok, let's continue

- In `makewandef3d.py` change the energy window to `emax=10`.
- Execute the script.

```
python makewandef3d.py
```

- Re-run fplo. (If you followed out setup advise, you only need to hit `enter` in the terminal running fplo.)
- When this is done hit `Ctrl-R` in xfbp to obtain the left panel of Figure 3:

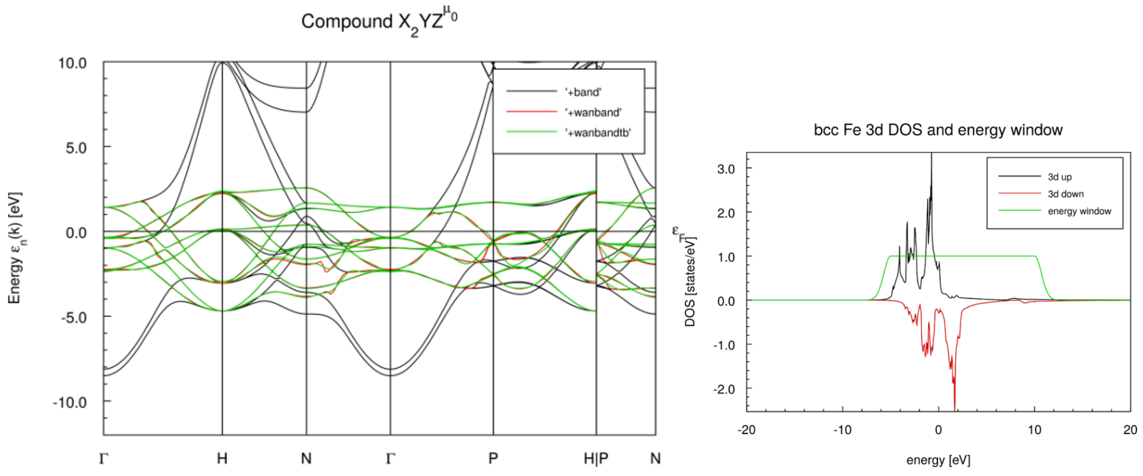

Figure 3: band, wanband and wanbandtb: larger energy window.

Well, this looks better except for a few wiggles. which again come from a rather abrupt cutoff at 10eV. What can remedy this? A smooth cutoff.

- change the upper fall-off to `deupper=7`.
- execute the script.

```
python makewandef3d.py
```

- Re-run fplo.
- and when this is done hit `Ctrl-R` in xfbp to obtain the left panel of Figure 4:

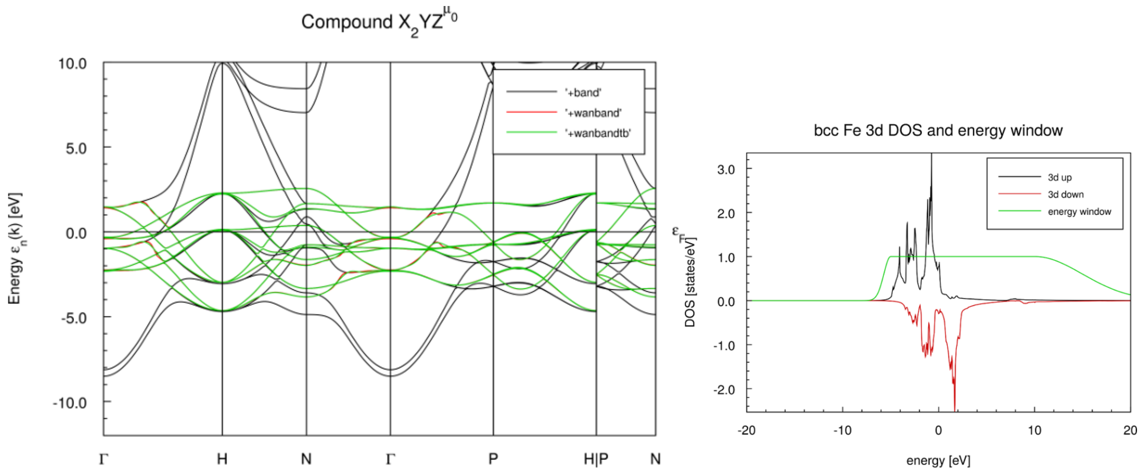

Figure 4: band, wanband and wanbandtb: larger energy window and smooth upper tail.

This is even better, but still wiggly and most of all the bands get pulled to higher energies between PH and PN (not surprising).

- change the energy window to `emax=0`.
- execute the script.

```
python makewandef3d.py
```

- Re-run fplo.
- and when this is done hit `Ctrl-R` in xfbp to obtain the left panel of Figure 5.

  Why does that work. The large upper tail covers what we need form higher bands. But it also covers the main 3d-part even if we put the upper window boundary (`emax`) below the main part. Moreover, it will pull the bands back down, since the main weight from the window function shifts down. As a side comment, we cannot choose different energy windows for spin up and spin down, since we have a full relativistic calculation.

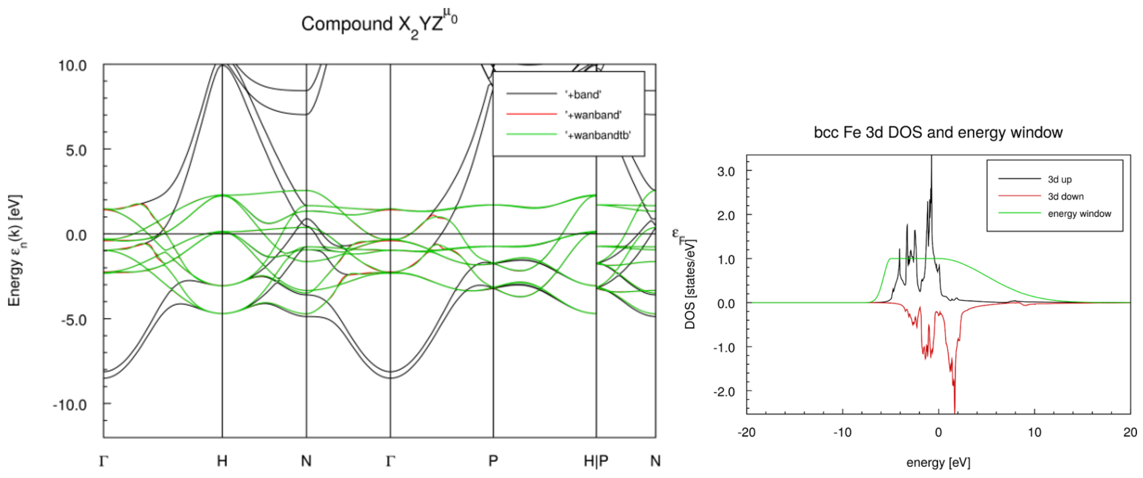

Figure 5: band, wanband and wanbandtb.

Yet the wiggles are still there. Any idea? It is the lower boundary of the energy window, which is still too "discontinuous". Afterall, a free-electron band comes in from below.

- change the lower fall-off to `delower=7`.
- execute the script.

```
python makewandef3d.py
```

- Re-run fplo.
- and when this is done hit `Ctrl-R` in xfbp to obtain the left panel of Figure 6

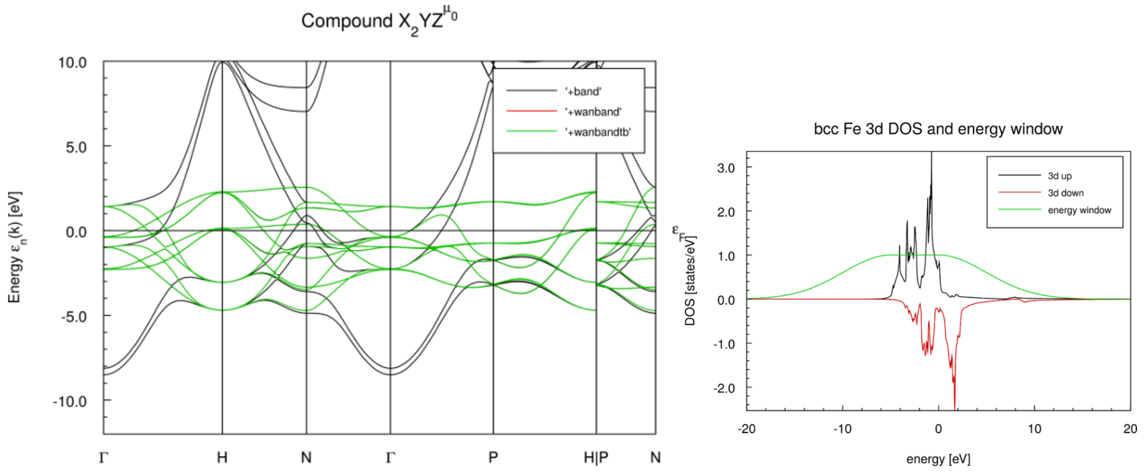

Figure 6: Magic: band, wanband and wanbandtb.

Now, one could play with the energy window parameters a bit more to optimize. But, this is already a very good choice. The WF fit is not perfect (free-electron band problem and some deviations) but surprisingly good.

- The energy window must have large tails.
- The core-window can be rather narrow, in fact it could be `emin=emax` in many cases (also in ours).
- Shifting the window around pulls the resulting WF bands higher or lower.

We will now draw your attention to the localization aspect.

- Please execute

```
xfbp WFstat.xpy
```

Right click on the legend box and chose `hide` to get Fig. 7. The picture shows the magnitude of the contributions of individual orbitals to the WFs as a function of the distance of said orbitals to the WF center on a logarithmic plot. In other words it depicts the WF extend. You will notice that the WF is surprisingly localized. In former versions one could the these curves rising again at larger distances due to some replica induced by the finite SCF k-mesh, according to the Nyquist–Shannon sampling theorem. Since Version 19, we use a routine, which eliminates such replica in a symmetric way. This leads to better agreement between the infinitely accurate transform (`+wanband`) and the cut-off version, which leads to the actual real space WF model (`+wanbandtb`). This new treatment also means that the number of possible WF matrix elements is always finite, although much larger than obtained for our current cut-off.

– Please execute

```
grep -4 R-vectors outwf
```

to get

```
(Number of R-vectors        869 Number of k-vectors      1728
R-mesh not yet at maximum possible size. -> Increase ham_cutoff(rcutoff)
```

This tells us that we use 869 real space vectors around a WF center for the model. The maximum possible number equals the number of k-vectors. Increasing the rcutoff will of course lead to a slower calculation and a larger +hamdata file.

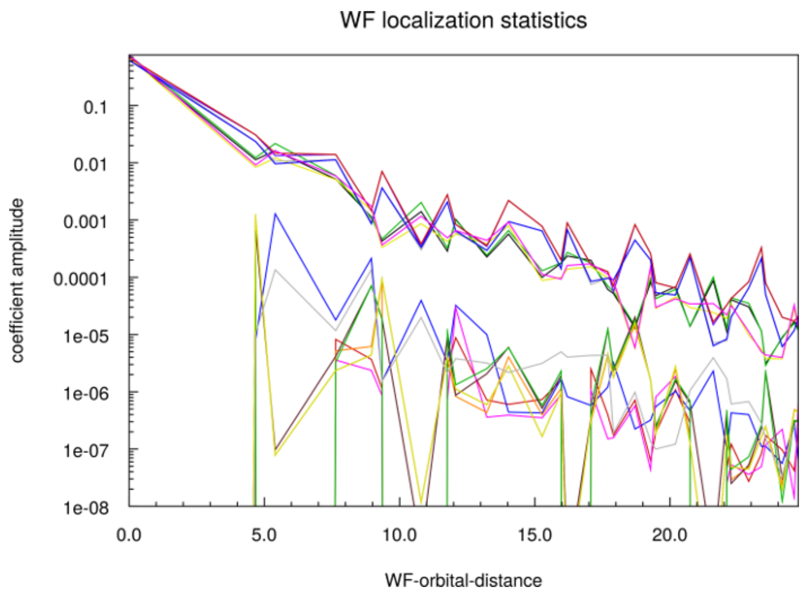

Figure 7: The extend of the WF orbital contributions.

3 Minimal basis bcc-Fe Wannier functions.

For practical applications (pyfplo.slabify) we want a model, which describes all bands around the Fermi energy well. To achieve this we need to add the 4s and 4p orbitals. The 4p orbitals are needed e.g. for the yellow spot at the N-point in Fig. 1.

- Copy the script

```
cp makewandef3d.py makewandef.py
```

- Edit makewandef.py and in line 20 add ,'4sb','4pb' to the orbital list. Set emin to -9 and delower to 1. The latter is obviously reasonable since we fit the whole set of occupied bands.

- execute the script.

```
python makewandef.py
```

- Re-run fplo and **continue reading this document**!

- Oops!

  What happend is that the energy window is way too small to encompass a sufficient amount of sp weight in the unoccupied bands. There are sections in the BZ where the total amount of sp-weight is zero or nearly zero. This leads to a deletion of Hilbert space vectors such that the resulting projected basis is deficient.

- Edit `makewandef.py` and put `deupper=10`. Also add `printT=False,` (including the final coma!) after `savespininfo=True,`. This will switch off the output of the lines `T=...` in the output. After all, this information is contained in a much more useful manner in `+hamdata`.

- execute the script.

  ```
  python makewandef.py
  ```

- Re-run fplo.

- No oops, but SVD (singular value decomposition) warnings. We still have a problem.

- In the still running xfbp open the `Edit ▷ Script/Transformations` dialog and change the value in the last line from 10 to 35.

- Save the script and hit `Apply` or close the dialog and hit `Ctrl-R`. You will see that the bands needed for the fit reach up to 35 eV.

- Edit `makewandef.py` and put `deupper=15`

- execute the script.

  ```
  python makewandef.py
  ```

- Re-run fplo.

- and when this is done hit `Ctrl-R` in xfbp to obtain the left panel of Fig. 8 (We rescaled the figure, yours shows a larger window.):

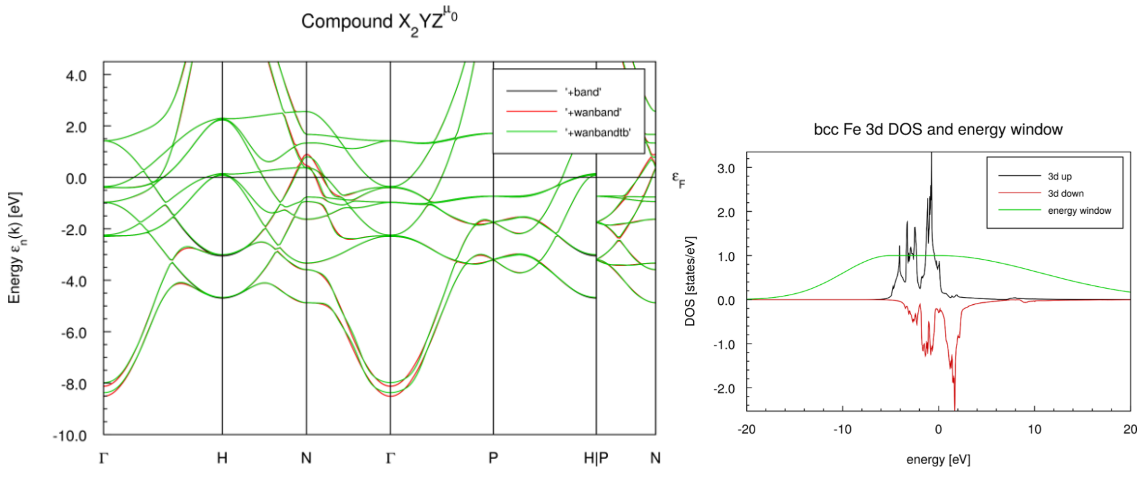

Figure 8: Minmum basis WFs.

You will see in Fig. 8 that the tight-binding approximation (green) follows the exact WF-transformation (red) quite closely except for around N and at the band bottom. The latter problem has to do with the fact that we approximate nearly free electron bands by tight binding. This is similar to a Fourier decomposition. One needs a number of frequencies to obtain a good approximant. In our case we can increase the Hamilton `rcutoff` to 30.

- Edit `makewandef.py` and put `rcutoff=30`
- execute the script.

```
python makewandef.py
```

- Re-run fplo and when this is done hit `Ctrl-R` in xfbp to obtain Fig. 9. Note, that the calculation will take a longer time now. This is mostly due to the use of symmetries. For larger cells, with less symmetry operations, this overhead reduces. In other words the time for this step in the calculation does not scale up linearly with the number of sites.

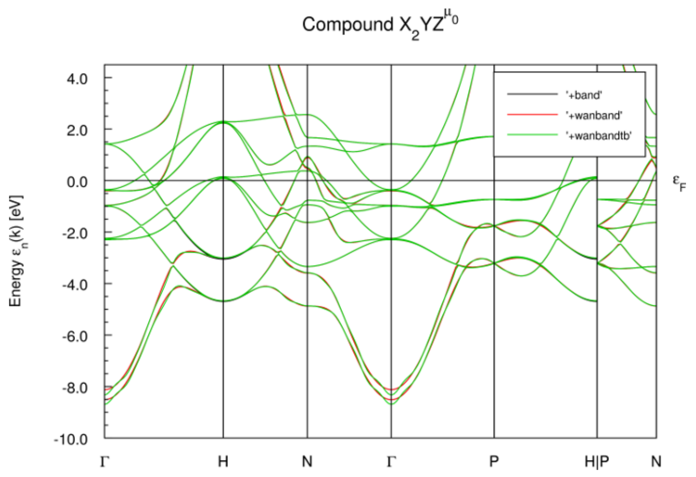

Figure 9: Larger cutoff.

You will see that there is an issue at the band bottom, which has to do with the WF extend.

- Execute

```
xfbp WFstat.xpy
```

and hide the legend box. You will see:

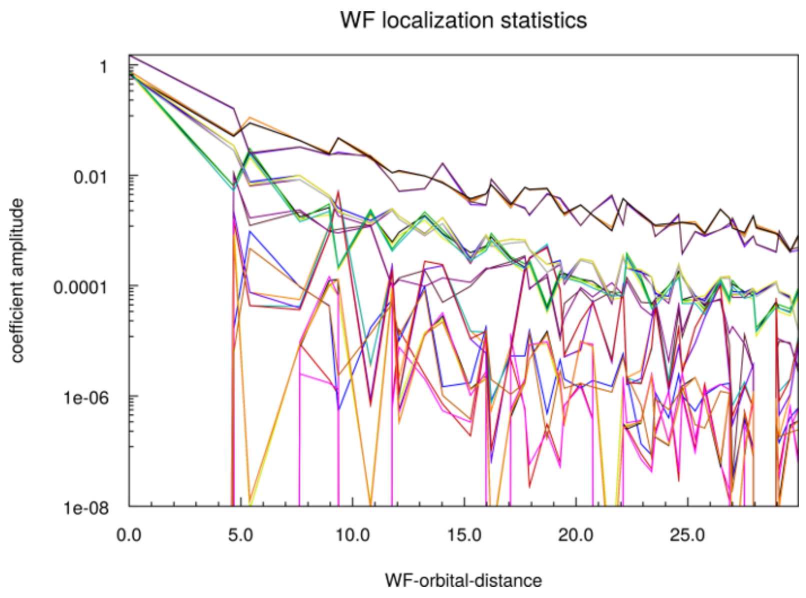

Figure 10: The extend of the WF orbital contributions.

This is now less localized than for the 3d-only case in Fig. 7 because of the inclusion of the more extended 4s- and 4p-orbitals.

- Please grep the R-vector information:

  ```
  grep -4 R-vectors out
  ```

  and you will see

  ```
  (Number of R-vectors        1407 Number of k-vectors       1728
  R-mesh not yet at maximum possible size. -> Increase ham_cutoff(rcutoff)
  ```

  We still do not use all possible vectors.

Just to see what happens, let's increase the `rcutoff` further.

- Edit `makewandef.py` and put `rcutoff=40`
- execute the script.

  ```
  python makewandef.py
  ```

- Re-run fplo and when this is done hit `Ctrl-R` in xfbp to obtain Fig. 9

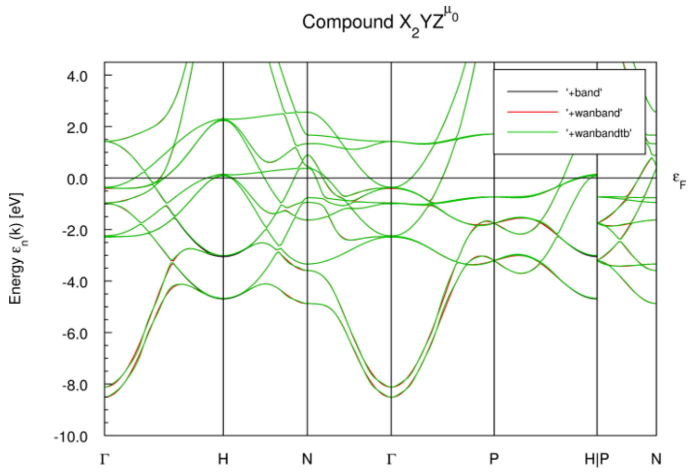

Figure 11: More k-points.

- Please execute one more grep for the the R-vector information:

```
grep -4 R-vectors out
```

and you will see

```
(Number of R-vectors is at maximum Number of k-vectors      1728
```

We finally exhausted all possible R-vectors and as can be seen in Fig. 11, the Problems at the band bottom are gone. A check of the extend

```
xfbp WFstat.xpy
```

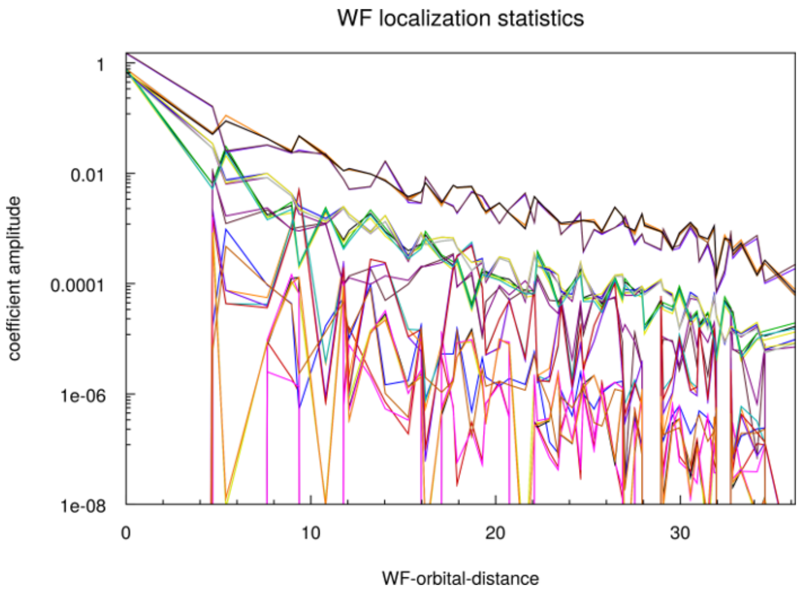

Figure 12: The extend of the WF orbital contributions.

reveals that the orbital contributions fall off to 10^{-4} at most.

Please note, that one does not always need to extend the rcutoff to its maximum. In our case this is required by the extended nature of the free electron parts and by the disentangling situation. There are many cases when a rcutoff of 30 seems to be enough.

If you zoom into the band bottom you will realize that the green bands (from the finite model) oscillate around the exact bands. This is an error which comes from the finite size of the SCF k-mesh, which limits the number of Fourier components. To remedy this we will use a larger k-mesh:

- So, call fedit and set number of iterations to 1 and k-mesh subdivision to 16,,
- Remove the WF data

```
rm +wancoeff +wancoeffbin
```

- Completely re-run fplo: First interrupt the running fplo: Ctrl-C and re-run it to re-create `+wancoeff`.
- Then re-run once more to re-calculate the WFs.

  You get

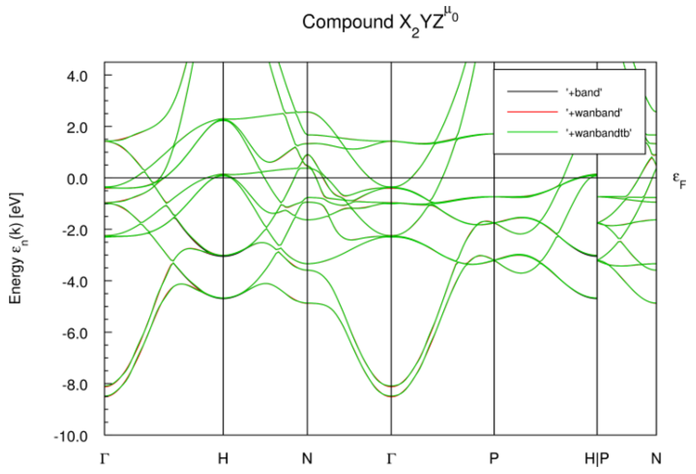

Figure 13: More k-points.

  We still have some tiny problems at some places, e.g. the lowest two bands at `H|P` are a bit too high.

- So, let's reduce the upper boundary a bit: edit `makewandef.py` and put emax=-1
- execute the script.

```
python makewandef.py
```

- Re-run fplo

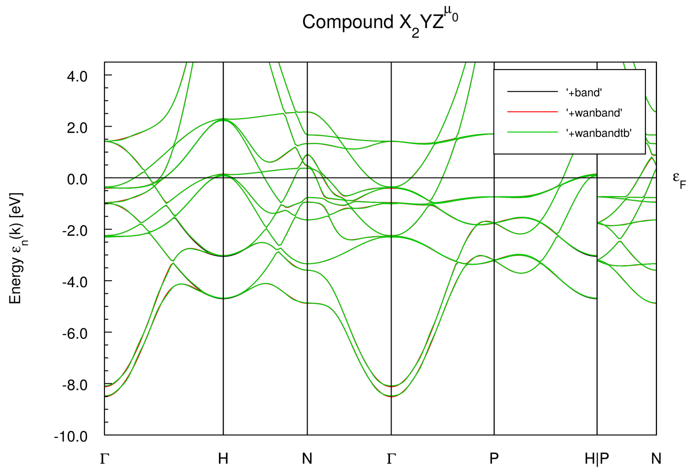

Figure 14:

When you grep the R-vector information you will notice that with the larger k-mesh the possible set of R-vectors also increased. One could further increase the `rcutoff` and play with the energy window to optimize tiny deviations here and there. But around the Fermi level our fit is good enough for our purpose.

4 Bulk-projected bandstructure

- Now, copy some prepared files and change into `slabify`.

```
cp -r ../slabify .
cd slabify
```

- Execute

```
python bpb.py
```

- Study the script. Note, that the Fermi surface spectral function is much slower. That is due to the fact that the first plot has N_e energy points and N_k k-points, while the second plot has 1 energy point and N_k^2 k-points. Filling the energy axis for each k-point in the first plot requires basically one diagonalization, while in the second plot we need one diagonalization for each k-point and $N_k^2 \gg N_k$. To make it shorter we chose a rather course grid for the second plot.

- When the script is done execute

```
xfbp bpbedc.xpy
```

and

```
xfbp bpbfs.xpy
```

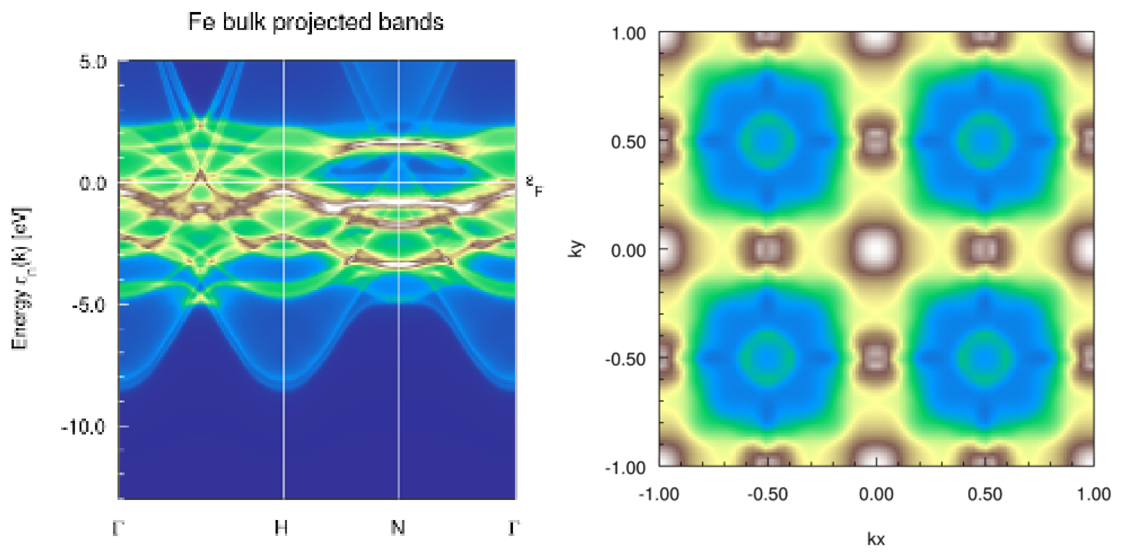

Figure 15: Bulk projected bands and fermi surface.

5 Fermi surface cuts.

The Wannier function model (`+hamdata`) can also be used to calculate Fermi surface cuts.

- Inside the `slabify` diretory execute

  ```
  python fscuts.py
  ```

- Study the script.
- When the script is done execute

  ```
  xfbp fscuts.xpy
  ```

  You will get Fig. 16. Also study the xfbp script (`fscuts.xpy`).

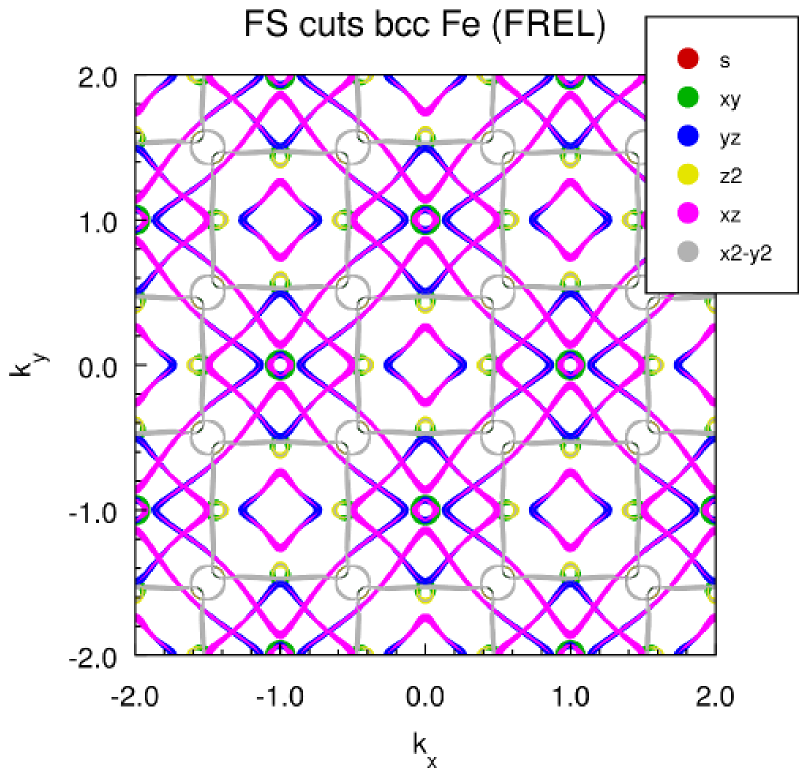

Figure 16: Fermi surface cuts colored by band weights.