**PyProcar Manual**

**A python package for vasp and abinit post-processing.**

**Contributors:**

Francisco Munoz

Aldo Romero

Sobhit Singh

Uthpala Herath

Pedram Tavadze

This code reads the PROCAR file obtained from VASP or Abinit calculations and does the post processing.

The following plots can be obtained.

1. the electronic bandstructure along a defined *k*-path,
2. the spin-texture calculated at a constant energy surface in a 2D *k*-mesh,
3. Orbital/Atom/Spin projected bandstructure,
4. Fermi-surface, (Pedram,Uthpala: *TBD)*
5. Optical transition matrix elements in the full Brillouin zone (*TBD*).

* OUTCAR file or an equivalent Abinit output file is required to extract the Fermi-energy and reciprocal lattice vectors.*

**Note**: We need to first repair the PROCAR file before it can be parsed by the PyProcar code. The reason for this is the improper writing (without space) of some k-points in the VASP-generated PROCAR file. For example: **0.33333-0.3333-0.00000**

To start off, import pyprocar as follows.

* import pyprocar

**To repair PROCAR, use the following command:**

* pyprocar.repair(‘PROCAR’, ‘PROCAR-repaired’)

Now use the PROCAR-repaired file for further processing of data.

**To quickly plot the electronic bandstructure:**

* pyprocar.bandsplot(‘PROCAR-repaired’, outcar=’OUTCAR’)

**To set Fermi-level at a particular value [e.g. EF = 0.1 eV]:**

* pyprocar.bandsplot(‘PROCAR-repaired’, outcar=’ OUTCAR’,**fermi= 0.1)**

**To set limit along energy axis [e.g. -2.0 to +2.0 eV]:**

* pyprocar.bandsplot(‘PROCAR-repaired’, outcar= ‘OUTCAR’,**elimit= [-2 2])**

***Different features of matplotlib can be utilized to make desired plot. Some are given below for example:***

* pyprocar.bandsplot(‘PROCAR-repaired’,outcar=’OUTCAR’,elimit=[-2 2],**mode=’parametric’)**

*“mode=parametric”* is used to plot lines instead of points by interpolation of data. This may induce small artifacts near the band-crossing.

Other options of matplotlib can also be used. One can define the color map by setting the argument “*cmap*”. For example:

* pyprocar.bandsplot(‘PROCAR-repaired’,outcar=’OUTCAR’,elimit=[-2 2], mode=’parametric’,**cmap=’seismic’])**

**Note:** For spin polarized plot of bands you need a two colors scheme (*e.g.* ‘seismic’ for red and blue bands)

*Other options:*

* pyprocar.bandsplot(‘PROCAR-repaired’,outcar=’OUTCAR’,elimit=[-2 2],**kticks=[0 49 99], kticksnames=[‘M’, ‘G’, ‘K’],cmap=’seismic’, vmin=-1,vmax=1,**mode=’parametric’])

elimit: energy range to plot

kticks, kticksnames: points and name of the special k-points. Keep in mind 50th k-point is #49 because counting starts from 0.

vmin, vmax: To define max. and min. value for projection

**For spin-projection:**

For Stotal, Sx, Sy, and Sz components of spin in a ***non-collinear calculation*** use spin=0, 1, 2, 3 (respectively). For example:

* pyprocar.bandsplot(‘PROCAR-repaired’,outcar=’OUTCAR’,elimit=[-2 2],kticks=[0 49 99], kticksnames=[‘M’, ‘G’, ‘K’], cmap=’seismic’, vmin=-0.5,vmax=0.5,mode=’parametric’,spin=**1)**

This feature is particularly useful to analyze the Rashba-Dresselhaus effect.

**For orbital-projection:**

For orbital projections on bands use ‘orbitals’. *For example*: for p*z* orbital-

* procar.bandsplot(‘PROCAR-repaired’,outcar=’OUTCAR’,elimit=[-2 2],kticks=[0 49 99],kticksnames=[‘M’, ‘G’, ‘K’],cmap=’seismic’,vmin=-0.5,vmax=0.5,mode=’parametric’, orbitals= **3)**

The counting goes as follows (as per the ordering of orbitals in the VASP’s PROCAR file):

**s py pz px dxy dyz dz2 dxz dx2**

**0 1 2 3 4 5 6 7 8**

**To filter only selected bands which are close to the fermi-level:**

* pyprocar.filter( ‘PROCAR-repaired’, ‘PROCAR-repaired-band50-80’,bands=[50 80])

This feature is very usefully when there are many bands in the PROCAR file (e.g. in heterostructures or supercells) and we are interested only in those bands that are near the Fermi-level. We can check the band indices near the Fermi-level from the OUTCAR file (*hint*: check occupancy), and filter only the bands in the vicinity of the Fermi-level using tag ‘filter –b <band\_first> <band\_last>’. This reduces the PROCAR file size, thus the memory requirements for post-processing of data. Now, the new PROCAR file (PROCAR-repaired-band50-80) can be used for the post-processing.

**To filter only selected orbitals and make a new PROCAR:**

* pyprocar.filter(‘PROCAR’,’PROCAR-REPAIRED\_sp’, **orbitals=[[0],[1,2,3] ])**

It will make a new PROCAR file containing only three columns, one for *s* orbitals, one for *p,* and one for total (*real total, not total of s+p*).

If you want to select only *px* orbitals, just use “orbitals= **3**”.

In the same way to plot the projection of total *p*-orbitals use “**orbitals=**” followed by 1+2+3 (i.e. *px+py+pz*). Same goes for other orbitals.

**To select atoms:**

* pyprocar.filter(‘PROCAR’,’PROCAR-REPAIRED\_ATOMS’, atoms=**0)**

*procar.py* counts each row of ions in PROCAR file starting from 0. If you just want to plot the contribution of first ion, use “**atoms**” followed by 0, to plot contribution of all same kinds of atoms, use 0+1+2. (here I am considering three atoms of same type in the POSCAR). Keep in mind: “atoms=**0**” does not define the *atom\_type*, rather it defines only the *first ion* in the POSCAR. So if you have more than one ion of the same element, use “**–a 0+1+2+3+**….”

**To plot the spin-texture:**

* **pyprocar.fermi2D(‘PROCAR-repaired’,**outcar=’OUTCAR’,energy=-1.0,st=True)

*Details:*

fermi2D: calls for 2D spin-texture plot

energy: to set the constant energy value

**To plot spin-texture for Sy component without using arrows:**

* **pyprocar.fermi2D(‘PROCAR-repaired’,**outcar=’OUTCAR’,energy=-1.0,st=True,noarrow=True, spin=**2)**

**Translate and Rotate the 2D KPOINT mesh:**

For any spin-texture in 2D K-plane, pyprocar usually treats Z-direction as the normal, and makes plot in the X-Y plane. It works fine when we have a 2D k-mesh in (Kx, Ky, 0) plane, but for other 2D k-meshes (*e.g.* Y-Z or X-Z), it gives us 'Value Error' or 'Segmentation Fault'.

**Solution** is to rotate the PROCAR by 90 degrees and make the Z-axis perpendicular to the 2D plane. But we should first translate our mesh to a particular K-point (which is mostly the center of the k-mesh) to define the rotation point, and then decide the rotation axis and rotation angle.

To translate the k-mesh: translate=[**2 2 0] ## (***220 is the index of the k-point in the 2D k-mesh, counting starts from 0)* to define the rotation point

To define the rotation angle and rotation axis: rotation=[90 0 1 0] ## This defines the rotation angle = 90, and Y-axis as the rotation axis.

procar always performs the translation operation first and then does the rotation.

**Be careful**: Now *Sx*, *Sy* and *Sz* all will be rotated according to the user’s choice. *S*y will be same as before but *Sx* would now be *Sz* and *Sz* would be -*Sx*.

Mergeabinit

Used to merge multiple PROCAR files from parallely run Abinit.

Usage:

>pyprocar.mergeabinit(‘PROCAR\_merged’)

generate2dkmesh

Used to generate a k-mesh for obtaining Fermi surfaces.

Usage:

>pyprocar.generate2dkmesh(x1,y1,x2,y2,grid\_size)

kpath

Used to generate a KPOINTS file with paths for band structure calculation.

Usage:

>pyprocar.kpath(infile,grid\_size,with\_time\_reversal,recipe,threshhold,symprec,angle\_tolerence)

Ex:

**>pyprocar.kpath('POSCAR',40,True,’hpkot’,1e-07,1e-05,-1.0)**

The POSCAR should be in the following standard format:

Sb Bi

4.51004000000000

0.8660254037844390 -0.5000000000000000 0.0000000000000000

0.0000000000000000 1.0000000000000000 0.0000000000000000

0.0000000000000000 0.0000000000000000 2.6420852143218241

3 3

Direct

0.0000000000000000 0.0000000000000000 0.6470799999999988

0.6666666666666643 0.3333333333333357 0.9804133333333345

0.3333333333333357 0.6666666666666643 0.3137466666666702

0.0000000000000000 0.0000000000000000 0.1818699999999997

0.6666666666666643 0.3333333333333357 0.5152033333333354

0.3333333333333357 0.6666666666666643 0.8485366666666640

0.00000000E+00 0.00000000E+00 0.00000000E+00

0.00000000E+00 0.00000000E+00 0.00000000E+00

0.00000000E+00 0.00000000E+00 0.00000000E+00

0.00000000E+00 0.00000000E+00 0.00000000E+00

0.00000000E+00 0.00000000E+00 0.00000000E+00

0.00000000E+00 0.00000000E+00 0.00000000E+00

**pyprocar.cat()**

>pyprocar.cat([‘PROCAR1’,’PROCAR2’],’PROCAR\_merged’)