Computer Physics Communications 251 (2020) 107080

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

PyProcar: A Python library for electronic structure pre/post-processing☆,☆☆

Uthpala Herath [a], Pedram Tavadze [a], Xu He [b], Eric Bousquet [b], Sobhit Singh [a,c], Francisco Muñoz [d,e], Aldo H. Romero [a,*]

[a] Department of Physics and Astronomy, West Virginia University, Morgantown, WV 26505-6315, USA
[b] Physique Théorique des Matériaux, CESAM, Université de Liège, B-4000 Sart-Tilman, Belgium
[c] Department of Physics and Astronomy, Rutgers University, Piscataway, NJ 08854, USA
[d] Departamento de Física, Facultad de Ciencias, Universidad de Chile, Santiago, Chile
[e] Center for the Development of Nanoscience and Nanotechnology (CEDENNA), Santiago, Chile

ARTICLE INFO

Article history:
Received 14 June 2019
Received in revised form 7 November 2019
Accepted 18 November 2019
Available online 27 November 2019

Keywords:
DFT
Bandstructure
Electronic properties
Fermi-surface
Spin texture
Python
Condensed matter

ABSTRACT

The PyProcar Python package plots the band structure and the Fermi surface as a function of site and/or s,p,d,f - projected wavefunctions obtained for each k-point in the Brillouin zone and band in an electronic structure calculation. This can be performed on top of any electronic structure code, as long as the band and projection information is written in the PROCAR format, as done by the VASP and ABINIT codes. PyProcar can be easily modified to read other formats as well. This package is particularly suitable for understanding atomic effects into the band structure, Fermi surface, spin texture, etc. PyProcar can be conveniently used in a command line mode, where each one of the parameters define a plot property. In the case of Fermi surfaces, the package is able to plot the surface with colors depending on other properties such as the electron velocity or spin projection. The mesh used to calculate the property does not need to be the same as the one used to obtain the Fermi surface. A file with a specific property evaluated for each k-point in a $k-$mesh and for each band can be used to project other properties such as electron–phonon mean path, Fermi velocity, electron effective mass, etc. Another existing feature refers to the band unfolding of supercell calculations into predefined unit cells.

Program summary

Program Title: PyProcar
Program Files doi: http://dx.doi.org/10.17632/d4rrfy3dy4.1
Licensing provisions: GPLv3
Programming language: Python
Nature of problem: To automate, simplify and serialize the analysis of band structure and Fermi surface, especially for high throughput calculations.
Solution method: Implementation of a Python library able to handle, combine, parse, extract, plot and even repair data from density functional calculations. PyProcar uses color maps on the band structures or Fermi surfaces to give a simple representation of the relevant characteristics of the electronic structure.
Additional comments: Features: PyProcar can produce high-quality figures of band structures and Fermi surfaces (2D and 3D), projection of atomic orbitals, atoms, and/or spin components.
Restrictions: Only the VASP package is currently fully supported, the latest version of Abinit is partially supported (it will be fully supported in the Abinit versions 9.x). The PROCAR file format can easily be implemented within any DFT code.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
☆☆ The review of this paper was arranged by Prof. D.P. Landau.
* Corresponding author.
E-mail address: alromero@mail.wvu.edu (A.H. Romero).

https://doi.org/10.1016/j.cpc.2019.107080
0010-4655/

1. Introduction

Density Functional Theory (DFT) is one of the most widely used methodologies for electronic structure calculations in materials science [1–3]. Its very good quality/computational-cost ratio, together with the emergence of high performance computing, has reshaped the field of computational materials research. Decades efforts in developing DFT for simulation programs have led to highly efficient DFT codes which include, among others, Abinit [4], VASP [5,6], Siesta [7] or Quantum Espresso [8,9], which are capable of exploring remarkable material properties. Thorough calculations with these DFT codes have already been done for a wide range of compounds and the accumulated data can be publicly accessed from databases such as Materials Project [10], AFLOW [11], Materiae [12]. This new approach to materials has had a large impact into condensed matter physics developing new paradigms such as the quantum materials [13]. However, countless possibilities of exploring novel materials still exist and thus the necessity of efficient and reliable DFT pre- and post-processing tools.

As computational capabilities increase, the size and complexity of the systems under study increase as well, hindering the analysis and the capability of abstraction on key aspects of these systems. As physicists, we encountered this problem and realized that we require a tool designed to gain insight from our calculations but spending less time on coding. In particular, we have three requirements for such a tool: (i) the generation of a graphical representation of the quantity of interest – surface states, spin-texture, orbital projection, etc. – should take only a single line of input. This approach also enables the scripting of similar analysis, a must for high throughput calculations. (ii) We also realized the need of producing high-quality graphics, ideally using a vector format, suitable for further post-processing. (iii) Finally, there are several issues preventing a straightforward analysis of electronic structure results, *e.g.*, the size of output data files generated by DFT codes can be extremely large, thus making the post-processing of data very slow by excessive memory usage. In such a case, the post-processing tool, not the researcher, should be capable of taking care of large data files in a smart way and enabling the user to extract essential information while ignoring less interesting information from a large data file.

This paper focuses on the presentation of PyProcar, a robust, open source Python library used for pre- and post-processing of the electronic structure data coming from DFT calculations. PyProcar is capable of performing a multitude of tasks including plotting spin non-polarized and spin/atom/orbital projected band structures and Fermi surfaces — both in 2D and 3D, Fermi-velocity plots, unfolding bands of a supercell, comparing band structures from multiple DFT calculations and generating a k-path for a given crystal structure. Our code has matured throughout the last decade and currently has over sixty users who are members of the PyProcar forum [14]. To retrieve a reliable number of users of an open source code is a difficult task due to inconsistencies in keeping track of downloads or installations, nevertheless, according to Google BigQuery [15] PyProcar has been downloaded almost 9000 times and PyPI Download Stats [16] counts more than 100 installations of PyProcar per month based on the download analytics from the Python Packaging Index (PyPI) [17]. Even though, PyProcar is written to parse the PROCAR file from the VASP package (and from a fork of Abinit, which will be included in the next main production version), due to its object-oriented approach, PyProcar can be easily adapted to handle the output from other DFT codes. In Section 2 we will briefly explain some basic aspects of PyProcar. Then in Section 3 we present some examples illustrating the capabilities of the PyProcar code for analysis of the electronic structure.

2. Library overview

2.1. Electronic structure projection

The projection of the Kohn–Sham states over atomic orbitals and spins can give a large amount of essential information about the calculated system. However, such amount of information needs to be post-processed to extract the physical insight. In the VASP code, this information is written into the PROCAR file when LORBIT = 11 or 12 tag (to include phase projections of the wave functions) is specified in the INCAR, which arranges the information of projections in blocks, as seen in Fig. 1 This block is repeated if calculations are spin polarized and for noncollinear spin calculations, three additional such blocks are present which correspond to S_x, S_y and S_z spin directions. There also exists, a special format of PROCAR file that includes the phase information of the wave function, as described in Fig. 2, which is an important quantity required for band unfolding. The direction of the atomic orbitals (p_x, p_y, p_z, etc.) is defined with respect to the Cartesian coordinates x, y and z.

The site projected wave function in the PROCAR file is calculated by projecting the Kohn–Sham wave functions onto spherical harmonics that are non-zero within spheres of a Wigner–Seitz radius around each ion by:

$$|\langle Y^{\alpha}_{lm}|\phi_{nk}\rangle|^2$$

where, Y^{α}_{lm} are the spherical harmonics centered at ion index α with angular moment l and magnetic quantum number m, and $\phi_{n\mathbf{k}}$ are the Kohn–Sham wave functions. In general, for a non-collinear electronic structure calculation the same equation is generalized to:

$$\frac{1}{2}\sum_{\alpha,\beta=1}^{2}\sigma^{i}_{\alpha,\beta}\langle\psi^{\alpha}_{n,\mathbf{k}}|Y^{\alpha}_{lm}\rangle\langle Y^{\beta}_{lm}|\psi^{\beta}_{n,\mathbf{k}}\rangle$$

where σ^i are the Pauli matrices with $i = x, y, z$ and the spinor/ spin polarized wavefunction $\phi_{n\mathbf{k}}$ is now defined as

$$\phi_{n\mathbf{k}} = \begin{bmatrix}\psi^{\uparrow}_{n\mathbf{k}}\\ \psi^{\downarrow}_{n\mathbf{k}}\end{bmatrix}$$

This projection is performed for every k-point used in the DFT calculation, for every energy band and every atom in the unit cell. Output files can be stored as the absolute value of the projections, as it is usually needed for the band analysis or by also including the complex nature of the projections, which are used for the electron band folding. The raw data written on the PROCAR file captures most of the details that are essential to investigate interesting materials properties. But on the downside, these files are often too big to be analyzed without any additional computational support. For instance, a spin-polarized calculation involving 10 atoms, with 100 bands and 100 k-points gives almost 10^6 entries in the PROCAR file, and the file size grows with the number of atoms as $\mathcal{O}(n^2)$. Therefore, it is essential to reduce the PROCAR file size in an efficient way without losing the key information stored in the file. The PyProcar parser can be used to manipulate, reduce, filter, and comprehend the information stored in the PROCAR file.

```
PROCAR lm decomposed
# of k-points:  160         # of bands:   50         # of ions:    5
 k-point     1 :    0.00000000 0.00000000 0.00000000     weight = 0.00625000
band     1 # energy  -29.05608270 # occ.  2.00000000
ion      s     py     pz     px    dxy    dyz    dz2    dxz  x2-y2    tot
    1  0.972  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.972
    2  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    3  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    4  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    5  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
tot    0.974  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.974
band     2 # energy  -14.29383601 # occ.  2.00000000
ion      s     py     pz     px    dxy    dyz    dz2    dxz  x2-y2    tot
    1  0.011  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.011
    2  0.138  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.138
    3  0.234  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.234
    4  0.234  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.234
    5  0.234  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.234
tot    0.852  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.852
```

Fig. 1. Example of a PROCAR file structure for a spin non-polarized calculation. For spin polarized or noncollinear calculations there are additional blocks for each spin component. The PROCAR stores the real projections of the Kohn–Sham orbitals over the atomic orbitals in different blocks for each spin direction (x, y, z), as well as the total spin at each k-point in the Brillouin zone and for each electron band.

```
PROCAR lm decomposed + phase
# of k-points:  160         # of bands:   50         # of ions:    5
 k-point     1 :    0.00000000 0.00000000 0.00000000     weight = 0.00625000
band     1 # energy  -29.05624010 # occ.  2.00000000
ion      s     py     pz     px    dxy    dyz    dz2    dxz  x2-y2    tot
    1  0.972  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.972
    2  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    3  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    4  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
    5  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
tot    0.974  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.974
ion          s             py             pz             px             dxy
    1 -0.437 -0.883   0.000  0.000   0.000 -0.000  -0.000 -0.000   0.000 -0.000  ...
    2 -0.007 -0.015   0.000 -0.000   0.000 -0.000   0.000  0.000   0.000 -0.000  ...
    3 -0.009 -0.018  -0.000 -0.000   0.000  0.000  -0.000 -0.000   0.000  0.000  ...
    4 -0.009 -0.018  -0.000  0.000   0.000 -0.000   0.000 -0.000   0.000  0.000  ...
    5 -0.009 -0.018  -0.000 -0.000   0.000  0.000   0.000 -0.000   0.000  0.000  ...
charge 0.973          0.000          0.000          0.000          0.000          ...
```

Fig. 2. Example of a PROCAR file structure for a spin non-polarized calculation with a phase factor. This includes an additional block containing real and complex projections of the Kohn–Sham orbitals over the atomic orbitals.

2.2. PyProcar design

PyProcar is an object-oriented code, and despite its name, only the file-related classes (mostly the parsing) are related to the VASP's PROCAR and OUTCAR. The OUTCAR gives a summary of the calculation including information about the electronic steps, eigenvalues, Fermi energy, forces on the atoms, *etc*. PyProcar code parses the OUTCAR file and stores the Fermi energy, which is later used to shift the Fermi level to zero in band structure plots. The reciprocal lattice vectors are also parsed from OUTCAR. PyProcar is currently capable of parsing the outputs from a recent version of Abinit and given the code's flexibility, the extension to other codes is trivial. Once the data is stored in memory, PyProcar can be used regardless of the employed DFT code. PyProcar consists of several classes and functions, as mentioned below. The functions provide a high-level interaction with the user, allowing the relevant task to get done with minimum instructions. For convenience, and to allow an easy scripting, all these functions are fully decoupled. Also, the manipulation of the data can be saved/exported as a new file on the disk.

The low level work is carried by classes, each handling one specific task or sub-task, they are fairly complex and no interaction with the user is expected unless a new feature is needed. By design, the classes are as loosely connected as possible. Each class uses the `logging` module to provide a user-defined level of verbosity, which is very useful for debugging.

Fig. 3 displays an overview of the PyProcar library. PyProcar can be used to generate files required for DFT calculations such as a suitable KPOINTS file for both self-consistent and non self-consistent DFT calculations. The structures can be generated manually, or from one of many databases publicly available such as Materials Project [10], AFLOW [11], etc. Once the DFT calculation is complete, PyProcar uses the VASP generated outputs, `PROCAR` and `OUTCAR`, for further post-processing.

The core classes within PyProcar consist of `Procarparser`, `Utilsprocar` and `Procarselect` which parses the `PROCAR` data and stores them in organized arrays for later calculations or analysis. These core classes are explained briefly below.

1. `Utilsprocar`: This class contains modules to parse the `OUTCAR` file from a DFT calculation. It reads and stores the Fermi energy and the reciprocal lattice vectors. The Fermi energy is used to shift the Fermi level to zero in band structure calculations. The availability of the reciprocal lattice vectors enable `Utilsprocar` to convert the k-mesh grid between direct and Cartesian coordinates. The Fermi energy and reciprocal lattice vectors could also be provided manually through command-line inputs.

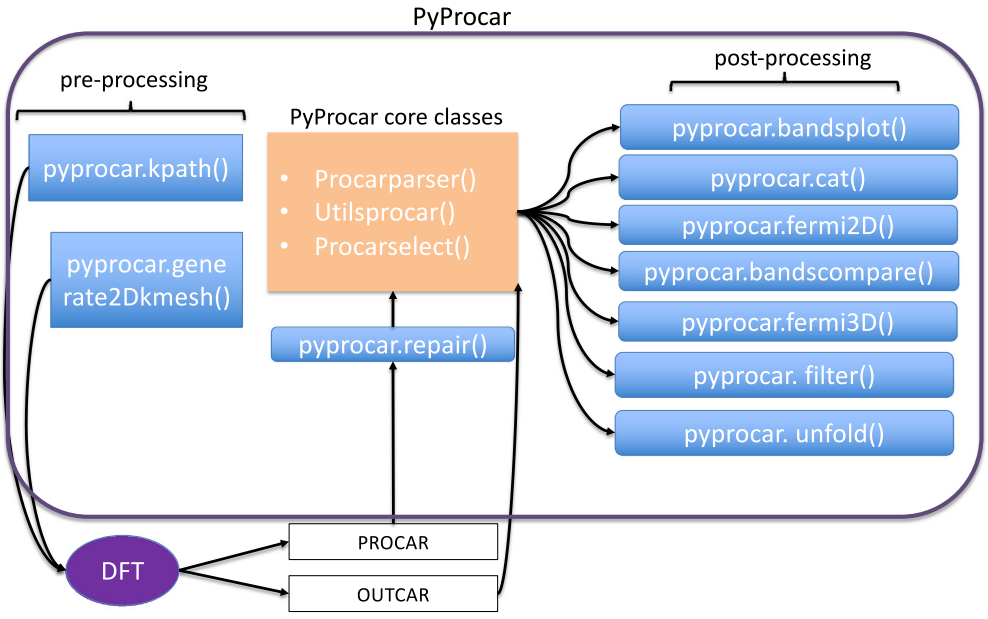

Fig. 3. A structural overview of the PyProcar library.

2. `ProcarParser`: As the name suggests, this class contains modules required to parse the `PROCAR` file. It reads information related to bands, *k*-points and orbitals, and saves the data in the memory. It is designed to be somewhat resilient to errors derived from the fixed-format of the `PROCAR` file (*i.e.,* missing blank spaces, use of *** characters as an index, etc.)
3. `ProcarSelect` After the `PROCAR` file is parsed and stored in memory, this class manipulates the required orbital(s), atom(s) and/or spin information separately.

2.3. Installation

The latest stable version of PyProcar, version 4.0.0 at the time of writing this paper, can be installed using the Python Packaging Index (`pip`) using the following command:

```
pip install pyprocar
```

The project's GitHub repository is located at https://github.com/romerogroup/pyprocar. An easy to follow documentation with examples can be found at https://romerogroup.github.io/pyprocar/. PyProcar is supported by both Python 2.x and 3.x.

3. Examples: MgB_2, BiSb, TaSb, and $SrVO_3$

3.1. The MgB_2, BiSb, TaSb, and $SrVO_3$ crystal structures

In this section, we choose four example systems, namely MgB_2, BiSb, TaSb, and $SrVO_3$, to illustrate the capabilities of the PyProcar code. Fig. 4 shows the crystal structure of these systems. The DFT calculations were performed using the Vienna Ab Initio Simulation Package (VASP) version 5.4.4. Similar calculations can be performed using the forthcoming version of Abinit, version 9.x, where the keyword *prtprocar* needs to be added in the input. The examples are available in the github repository.

For the strongly correlated cubic perovskite $SrVO_3$ (crystal symmetry $Pm\bar{3}m$ and lattice constant 3.84 Å), the plain and projected band structures were evaluated within the generalized gradient approximation (GGA) using PBE functional [18]. An $8 \times 8 \times 8$ Monkhorst–Pack [19] *k*-mesh and an energy cutoff of 600 eV were required for electron wavefunction convergence. We considered 10 valence electrons of Sr ($3s^2 3p^6 4s^2$), 5 valence electrons of V ($3d^3 4s^2$) and 6 valence electrons of O ($2s^2 2p^4$) in the PAW pseudopotential.

The superconducting material MgB_2 of the crystal symmetry $P6/mmm$ and lattice parameters 3.07 Å and 3.53 Å for *a* and *c*, respectively, was used to showcase the Fermi surface plotting and band unfolding capabilities of PyProcar. For the former, we used a $10 \times 10 \times 10$ Monkhorst–Pack grid with an energy cut-off of 700 eV within the GGA with the PBE exchange–correlation function. The latter was done with a $5 \times 5 \times 5$ Monkhorst–Pack grid with an energy cut-off of 500 eV within the GGA with the PBEsol exchange–correlation function [20]. Both of these calculations were done with 2 valence electrons of Mg ($3s^2$) and 3 valence electrons of B ($2s^2 2p^1$) in the PAW pseudopotential.

To demonstrate the 2D spin texture plotting capability of PyProcar we use the Rashba semiconductor BiSb monolayer which belongs to the space group $P3m1$ with lattice parameters $a = b = 4.26$ Å. The calculation, which included spin–orbit coupling (SOC) was performed with a *k*-mesh grid of $10 \times 10 \times 1$ and an energy cut-off of 650 eV with PBE using 15 valence electrons of Bi ($5d^{10} 6s^2 6p^3$) and 5 valence electrons of Sb ($5s^2 5p^3$) in the PAW pseudopotential.

PyProcar can also be used to investigate the band degeneracy and Dirac/Weyl points in topological materials. We use the material TaSb belonging to the space group $P\bar{6}m2$ with lattice parameters a = b = 3.58 Å, c = 3.81 Å to demonstrate this feature. An energy

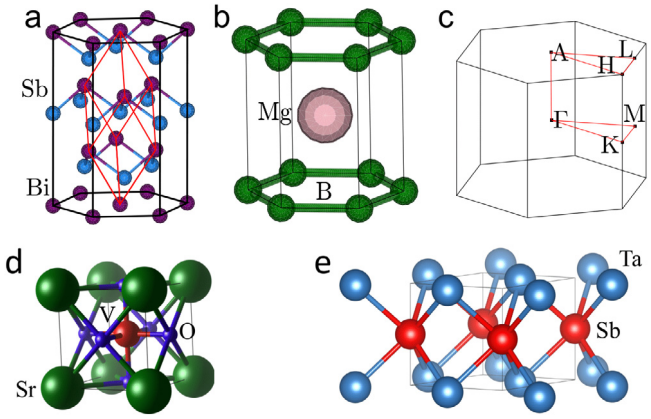

Fig. 4. (a) Crystal structure of a phase of BiSb with rhombohedral symmetry, the primitive cell is enclosed by red lines. (b) Hexagonal lattice of MgB_2. (c) First Brillouin zone of a hexagonal lattice. (d) Cubic lattice of $SrVO_3$. (e) Hexagonal lattice of TaSb. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

cut-off of 600 eV and k-mesh grid of $10 \times 10 \times 10$ along with PBE and SOC were used for the calculation considering 5 valence electrons of Ta ($5d^3 6s^2$) and 5 valence electrons of Sb ($5s^2 5p^3$) in the PAW pseudopotential.

3.2. Generating a k-path

In order to plot a band structure, one must define a set of k-points following a desired k-path in momentum space. A common option is to employ the AFLOW software framework [11], the SeeK-path module in Materials Cloud [21] or the Bilbao Crystallographic Server [22–24] to generate the k-path. However, PyProcar's k-path generation utility enables the user to automatically generate a suitable and sufficient k-path given the crystal structure, typically read from the `POSCAR` file. The k-path is then automatically written to a **KPOINTS** file. The k-path generation utility within PyProcar is based on the Python library `seekpath` developed by Hinuma et al. [25]. The red line in Fig. 4(c) depicts such a k-path of a hexagonal lattice generated using the algorithms of `seekpath`.

General format:

```
pyprocar.kpath(infile,grid_size,with_time_reversal,recipe,threshhold,symprec,angle_tolerence)
```

Usage:

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

More details regarding these parameters can be found in Ref. [26].

3.3. Repair

This utility is used to repair the ill-formatting of the PROCAR file due to the erroneous file handling in Fortran, particularly in a VASP calculation. This prevents issues arising from the lack of white space between a number and a negative sign, for instance `0.000000-0.5000000`. Typically, pyprocar.repair() is recommended to be applied before using any other utility.

Usage:

```
pyprocar.repair('PROCAR','PROCAR-repaired')
```

3.4. k-mesh generator

This utility can be used to generate a 2D k-mesh centered at a given k-point and in a given k-plane. The mesh is then automatically written to a **KPOINTS** file. This is particularly useful in computing 2D spin-textures and plotting 2D Fermi surfaces. For example, the following command creates a 2D $k_x - k_y$ mesh centered at the Γ point ($k_z = 0$) ranging from coordinates (−0.5, −0.5, 0.0) to (0.5, 0.5, 0.0) with 5 grids in the x direction and 7 grids in the y direction:

General format:

```
pyprocar.generate2dkmesh(x1,y1,x2,y2,z,num_grids_x,num_grids_y)
```

Usage:

```
pyprocar.generate2dkmesh(-0.5,-0.5,0.5,0.5,0,5,7)
```

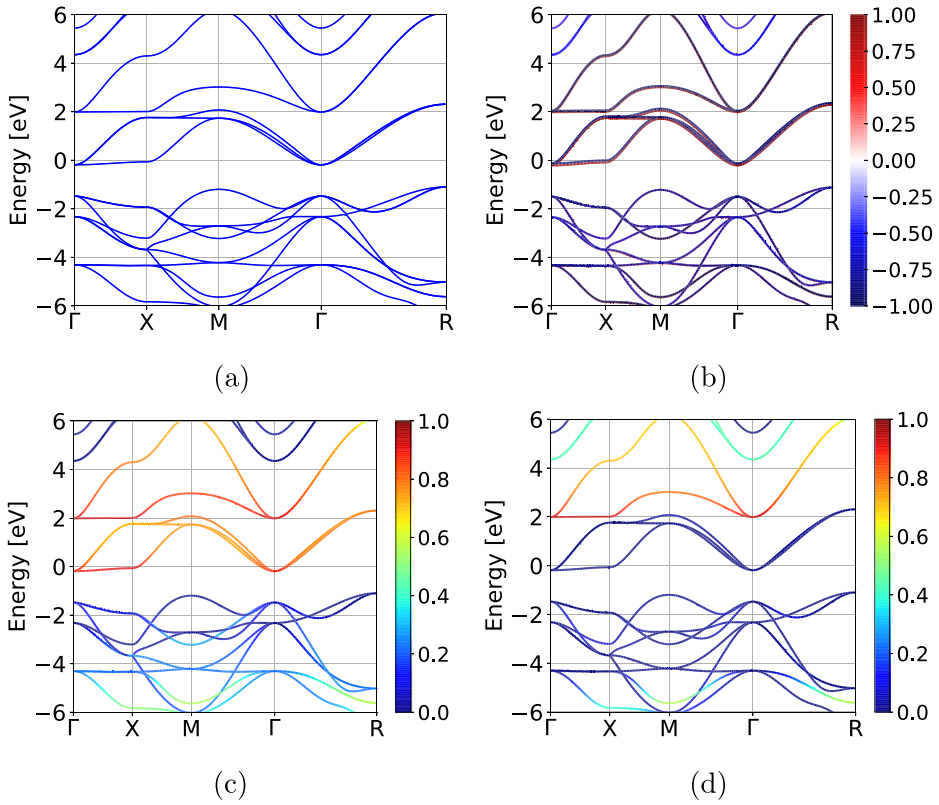

Fig. 5. (a) plain band structure (b) collinear spin projected (c) V atom projected (d) e_g orbital projected band structure of $SrVO_3$. In the projected plots, the color intensity corresponds to the degree of contribution of that particular orbital, spin or atom type. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.5. Band structure

PyProcar goes beyond the conventional plain band structure to plot the projected bands that carry even more information, which will be described shortly. The projected bands are color coded in an informative manner to portray fine details. PyProcar is capable of labeling the *k*-path names automatically, however, the user can manually input them as desired. This will be showcased in the next section.

1. **No projection**
   This is the most basic type of band structure. No projection information is contained here. See Fig. 5(a) for a plain band structure of $SrVO_3$. In order to use the plain mode one sets **mode** = 'plain'. **elimit** sets the energy window limits. **outcar** specifies the **OUTCAR** file. For Abinit calculations, **abinit_output** is used instead. **color** lets the user use any color available in the matplotlib [27] package.

   Usage:

```
pyprocar.bandsplot('PROCAR-repaired', outcar='OUTCAR', elimit=[-2,2], mode='plain', color='blue')
```

2. **Spin projection**
   For collinear spin polarized and noncollinear spin calculations of DFT codes, PyProcar is able to plot the bands of each spin channel or direction separately. An example for a collinear spin polarized calculation is given in Fig. 5(b) where blue corresponds to spin down channel and red to spin up channel. For this case setting **spin** = 0 plots the unpolarized spin density and **spin** = 1 plots the spin channels separately.

   Usage:

```
pyprocar.bandsplot('PROCAR_repaired', outcar='OUTCAR', elimit=[-5,5], kticks=[0,39,79,119,159], knames=['G','X','M','G','R'],
    cmap='seismic',
mode='parametric', spin=1)
```

   For noncollinear spin calculations, spin = 1, 2, 3 corresponds to spins oriented in S_x, S_y and S_z directions respectively. Setting **spin** = **'st'** plots the spin texture perpendicular in the plane (k_x, k_y) to each (k_x,k_y) vector. This is useful for Rashba-like states in surfaces. For parametric plots such as spin, atom and orbitals, the user should set **mode** = 'parametric'. **knames** and **kticks** corresponds to the labels and the number of grid points between the high symmetry points in the *k*-path used for the band structure calculation. At the end of this section we explain how to retrieve this automatically. LaTeXentries, such as `r'\Gamma'` also can be used. **cmap** refers to the matplotlib color map used for the parametric plotting and can be modified by using the same color maps used in matplotlib. If spin-up and spin-down bands are to be plot separately, one may use the **pyprocar.filter()** function to create two PROCAR's for each case and plot them individually. Refer to the user manual for more information.

3. **Atom projection**
   The projection of atoms onto bands can provide information such as which atoms contribute to the electronic states near the Fermi level. PyProcar counts each row of ions in the PROCAR file, starting from zero. In the example of a five atom $SrVO_3$, the indexes of **atoms** for Sr, V and the three O atoms would be 1, 2 and 3, 4, 5 respectively. It is also possible to include more than one type of atom. See Fig. 5(c) for an example of V atom projected bands in $SrVO_3$.

   Usage:

```
pyprocar.bandsplot('PROCAR_repaired',outcar='OUTCAR',elimit=[-5,5],kticks=[0,39,79,119,159],knames=['G','X','M','G','R'],
    cmap='seismic', mode='parametric',atoms=[1])
```

4. **Orbital projection**
   The projection of atomic orbitals onto bands is also useful to identify the contribution of orbitals to bands. For instance, to identify correlated d or f orbitals in a strongly correlated material near the Fermi level. It is possible to include more than one type of orbital projection. Fig. 5(d) displays an orbital projected band structure of $SrVO_3$. The mapping of the index of orbitals to be used in **orbitals** is as follows (this is the same order from the PROCAR file, see Fig. 2).

s	p_y	p_z	p_x	d_{xy}	d_{yz}	d_{z^2}	d_{xz}	$d_{x^2-y^2}$	$f_{y(3x^2-y^2)}$	f_{xyz}	f_{yz^2}	f_{z^3}	f_{xz^2}	$f_{z(x^2-y^2)}$	$f_{x(x^2-3y^2)}$
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

   Usage: To project all five d-orbitals:

```
pyprocar.bandsplot('PROCAR_repaired',outcar='OUTCAR',elimit=[-5,5],kticks=[0,39,79,119,159],knames=['G','X','M','G','R'],
    cmap='seismic',mode='parametric',orbitals=[4,5,6,7,8])
```

If a KPOINTS file from a VASP calculation is present, PyProcar automatically retrieves **kticks** and **knames** and labels the band structure plots accordingly for any of the above cases and also for the `bandscompare()` function in Section 3.10. For example:

```
pyprocar.bandsplot('PROCAR_repaired',outcar='OUTCAR',mode='plain',kpointsfile='KPOINTS')
```

One or many of the above can be combined together to allow the user to probe into more specific queries such as a collinear spin projection of a certain orbital of a certain atom.

Different modes of band structures are useful for obtaining information for different cases. The four modes available within PyProcar are `plain`, `scatter`, `parametric` and `atomic`. The `plain` bands contain no projection information. The `scatter` mode creates a scatter plot of points. The `parametric` mode interpolates between points to create bands which are also projectable. Finally, the `atomic` mode is useful to plot energy levels for atoms.

These projected band structures play a vital role in revealing the physics of a system. For instance, the V^{4+} projected bands displayed in Fig. 5(c) correctly captures the contribution from the V$-3d$ electrons with O$-2p$ states from -6 eV to -1 eV . Furthermore, in Fig. 5(d) where e_g states are projected, one can clearly notice the octahedral crystal field splitting in $SrVO_3$ where t_{2g} states exist below e_g states, as expected.

3.6. 3D Fermi surface

PyProcar's 3D Fermi surface utility is able to generate Fermi surface plots projected over spin, atoms and orbitals or a combination of one or many of each. This utility is also capable of projecting external properties that are provided on a mesh grid in momentum space. This feature is useful when one wants to project properties that are not provided in a PROCAR file such as Fermi velocity, electron–phonon coupling and electron effective mass. We divide this section into three sub sections, plain Fermi surface, projection of properties from PROCAR and projection of properties from external file.

1. **Plain Fermi surface**
   The `ProcarParser` class provides eigenvalues on a momentum space mesh grid. In all of the 3D Fermi surface functions in PyProcar the eigenvalues are interpolated using Fourier transform. This type of interpolation is suitable for eigenvalues as they are periodic in nature. The points in the mesh grid is generated in reduced space and some of the points sampled by VASP might not be in the first Brillouin Zone (BZ). The points outside of the first BZ can be returned to the first BZ using the reciprocal lattice vectors. As transforming the points to the first BZ will cause the distortion in the shape of the iso-surface, it is better to add the points missing from the first BZ then remove the points lying in the second BZ. In order to identify these points, we first create the boundaries of the first BZ by creating the Wigner–Seitz cell from the reciprocal lattice vectors. The BZ is a convex hull created by the points located at the corner of the BZ. To figure out if a point is inside the first BZ or outside the following procedure is performed on each point. A point is added to the collection of the points on the BZ corners and a new convex hull is created, if the convex hull is the same as the original BZ, the point is located inside, and if the BZ is different, the point is located outside of the first BZ. Since this operation can be slow in a scripting language like Python, we use data parallelism. This task is offloaded to a number of worker processes, which can be defined by the user. After obtaining enough points we use Lewiner marching cubes algorithm provided by scikit-image package [28]. This function provides vertices and faces required to create the Fermi surface. The last step that is needed is to transform the coordinates of the vertices from the reduced coordinates to the Cartesian coordinates. To visualize the Fermi surface, we have provided four different plotting packages, mayavi [29], matplotlib [30], plotly [31] and ipyvolume [32] which can be chosen by the user. The figures shown here are produced using mayavi.

   Usage:

```
pyprocar.fermi3D(procar,outcar,bands,scale=1,mode='plain',st=False,**kwargs)
```

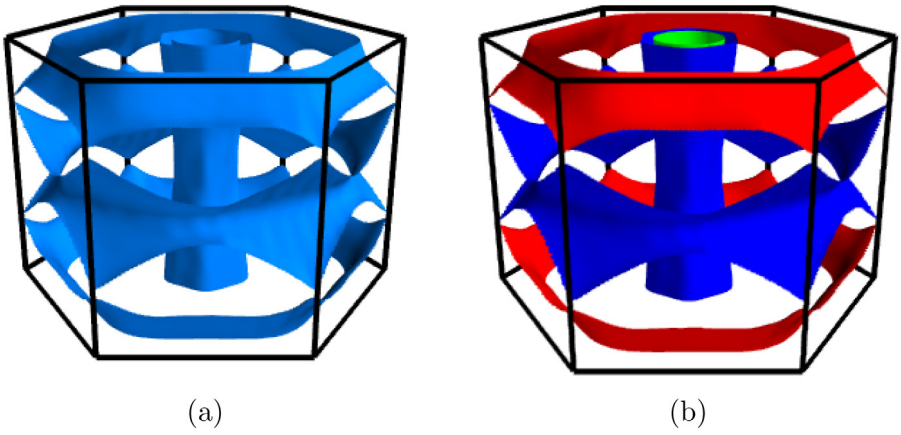

Fig. 6. (a) Plain 3D Fermi surface of MgB_2, (b) Plain 3D Fermi surface of MgB_2 with **face_colors** specified. Note that the face colors have to be provided in tuples of (r, g, b) normalized to 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

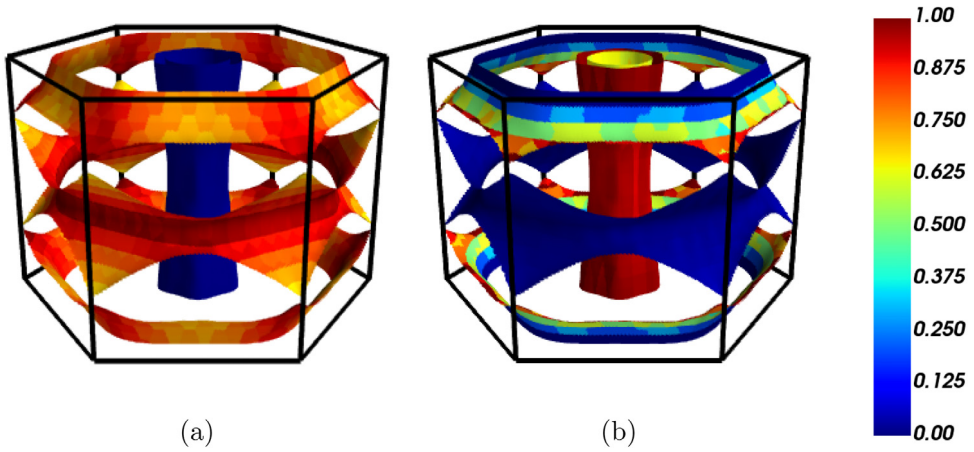

Fig. 7. (a) The projection of p_z orbitals of Boron atoms on Fermi surface of MgB_2 (b) Projection of p_x, p_y orbitals of Boron atoms on Fermi surface of MgB_2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The main arguments in this function are **procar**, **outcar**, **bands**, **scale**, **mode** and **st**, where **procar** and **outcar** are the names of the input PROCAR and OUTCAR files respectively, **bands** is an array of the bands that are desired to be plotted. Note if **bands** $= -1$, the function will try to plot all the bands provided in the PROCAR file. The k-mesh will be interpolated by a factor of **scale** in each direction. The **st** tag controls the spin-texture plotting, and **mode** determines the type of projection of colors. There are additional keyword arguments that can be accessed in the help section of this function, such as **face_color**, **cmap**, **atoms**, **orbitals**, **energy**, **transparent**, **nprocess** *etc.* Fig. 6 shows the Fermi surface of MgB_2 generated using `'plain'` mode of this function.

2. **Surface coloring based on properties from PROCAR**
Similar to the `bandsplot` section one can choose to project the contribution of different properties provided in the PROCAR file, such as atom, orbital and spin contributions. The projection can be represented by different color mapping schemes chosen by the user. The projection is not restricted to only one property at a time, so it can be chosen from all the provided properties. For example, one might want to see the contribution of the orbitals p_x, p_y, p_z from specific atoms, this function will parse the desired contributions and projects the sum of contributions on each face. To do so, we perform an interpolation on the data provided by the DFT package and evaluate the function at the center of each face created in the previous section. To use this functionality one has to change the **mode** from `'plain'` to `'parametric'` and choose the atoms, orbitals, spin that are desired to be projected. As an example, we project the different contribution of different p orbitals of Boron atom in Fig. 7. The results show the middle cylinder is mostly comprised of p_x and p_y orbitals while the bands closer to the edges of BZ are p_z orbitals. This is in agreement with the calculations performed using the SIESTA DFT package [33].
For noncolinear calculations, this function is able to plot arrows in the direction of the spinors provided in the PROCAR file. To turn this functionality on the one can set **`st`** $=$ `True` to turn the spin-texture ON. The user can choose between coloring all the arrows originated from one band with a specific color, or project the contribution of that arrow in a specific Cartesian direction. We plot two examples to demonstrate this functionality in Fig. 8, the spin-texture of BiSb at 0.60 eV above the Fermi level and the spin-texture of $SrVO_3$. To better represent the spin-texture we use the key argument **`transparent`** $=$ `True` which changes the opacity of the Fermi surface to zero.

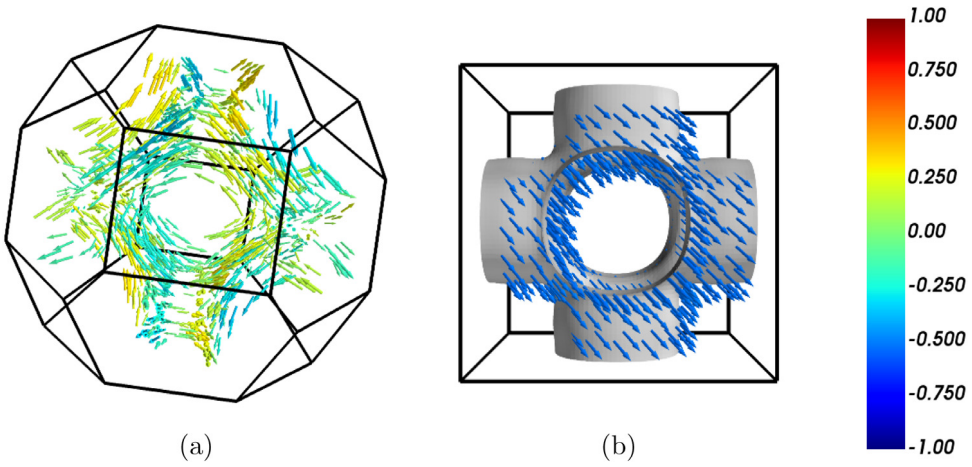

Fig. 8. (a) The spin-texture of BiSb calculated above the Fermi-level at $E = E_F + 0.60$ eV. One can clearly notice Rashba-type spin-splitting of conduction band electrons by analyzing spin-texture at $\mathbf{k}$ and $-\mathbf{k}$ wave vectors. (b) Spin texture of $SrVO_3$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. **Surface coloring based on properties obtained from an external file**
   Similar to the previous section, this function is able to read an external file, containing information about a scalar or a vector field in BZ and project the field on the Fermi surface. This file does not need to have the same mesh grid as the PROCAR file as long as the mesh sampling is fine enough. This function performs an interpolation on the provided data and evaluates functions at the center of each face on the Fermi surface. The external file should have the following format.

```
band = <band number>
  <kx1> <ky1> <kz1> <color1>
  <kx2> <ky2> <kz2> <color2>
  <kx3> <ky3> <kz3> <color3>
  ...
band = <band number>
...
```

   The function matches information about the first band present in the file to the first band requested to be plotted, second band present in the file to the second band requested to be plotted, and so on. As an example, we have plotted the Fermi velocity of MgB_2 calculated by numerically evaluating the energy gradient from the PROCAR file in Fig. 9. Of course, one can acquire more accurate Fermi velocity by means of the Wannier interpolation method [34], however the following plot shows close enough accuracy for an example designed to represent the capabilities of this functional mode.

3.7. Handling big files: filtering the selected data and reducing the memory requirement for post-processing

A simpler version of a PROCAR file containing only a subset of information from the original dataset can be generated with this utility. This feature is very useful when there are many bands in the PROCAR file (e.g. in heterostructures or supercells calculations) making the file size enormously large for post-processing while only bands near the Fermi-level are needed for analysis. In this case, one can filter data of selected bands near the Fermi-level. This considerably reduces the file size and makes the post-processing of data faster. For instance, in graphene/MoS_2 (5:4) heterostructures [35,36] the size of a VASP generated PROCAR file is 1.49 GB (98 atoms, 584 electrons, 804 bands in total, 63 k-points, and spin–orbit coupling included). However, by filtering only 30 bands above and below the Fermi-level (*i.e.* band indexes 554–614), the file size reduces to 108 MB and data processing becomes much faster. In the same way, one could use the `filter` utility to filter the `PROCAR` file to extract information regarding particular spins, atoms, or orbitals in a relatively smaller `PROCAR-new` file.

The following example extracts information of bands ranging from index 50 to 70 from a `PROCAR-repaired` file (Fermi-level is near band #60) while ignoring all other bands located far from the Fermi-level, and stores resulting dataset in a new file named `PROCAR-repaired-band50-70`. Now the new `PROCAR-repaired-band50-70` file can be used for further post-processing of data at relatively low memory requirements.

Usage:

```
pyprocar.filter('PROCAR-repaired','PROCAR-repaired-band50-70',bands=[50,70])
```

3.8. 2D spin-texture

This module can be utilized to visualize the constant energy surface spin textures in a given system. This feature is particularly useful in identifying Rashba and Dresselhaus type spin-splitting effects, analyzing the topology of Fermi surface, and examining Lifshitz

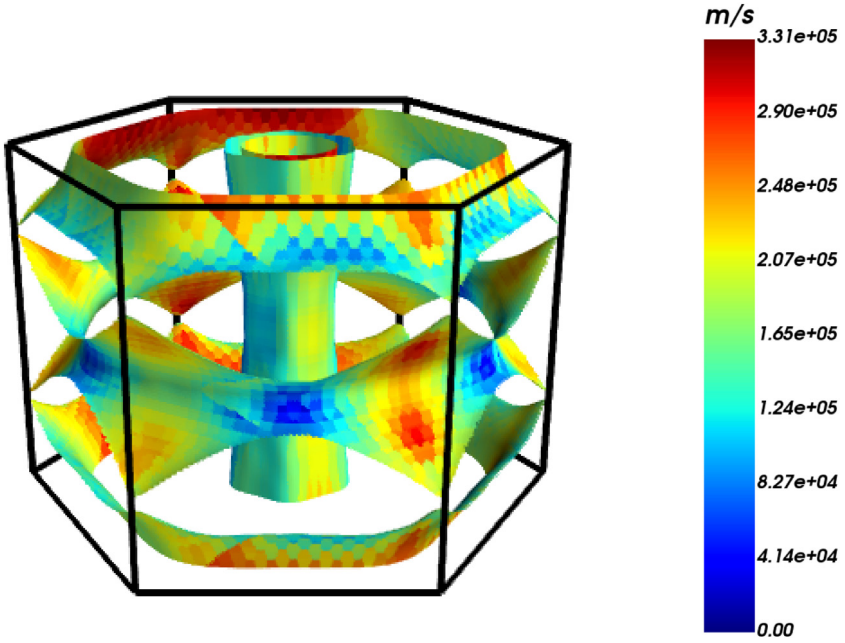

Fig. 9. Fermi velocity of MgB_2 projected on the Fermi surface. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

transitions, as demonstrated in Refs. [36–42]. To plot 2D spin texture, we require a 2D k-grid centered a certain special k-point in Brillouin zone near which we want to examine the spin-texture in k-space (see Section 3.4 regarding generation of 2D k-mesh).

Usage: To plot S_x spin component at a constant energy surface $E = E_F + 0.60$ eV (spin = 1, 2, 3 for S_x, S_y, S_z, respectively)

```
pyprocar.fermi2D('PROCAR-repaired', outcar='OUTCAR', st=True, energy=0.60, noarrow=True, spin=1)
```

For example, in Fig. 10 we plot S_x, S_y, and S_z spin projections on the 2D spin texture of BiSb monolayer, which is a Rashba semiconductor (Rashba spin-splitting takes place the Γ point of BZ), computed in a $k_x - k_y$ mesh centered at the Γ point. Fig. 10(a) shows spin-texture calculated above Fermi-energy (E_F) at constant energy value $E = E_F + 0.60$ eV (conduction bands), and Fig. 10(a) shows the spin-texture calculated below Fermi surface at constant energy value $E = E_F - 0.90$ eV (valence bands). One can notice linear in k Rashba spin-splitting effects in Fig. 10(a), and additional warping effects in Fig. 10(b) appearing due to the higher order k^3 terms in lower energy valence bands.

One could also plot spin texture using arrows instead of a heat map, as shown in Fig. 11. This can be done by setting tag: **noarrow** = False.

To set maximum and minimum energy values for color map, one could use `vmin` and `vmax` tags. For example, `vmin` = −`0.5`, `vmax` = `0.5` in Figs. 10 and 11.

3.9. Identification of Weyl points

Weyl points appear in the momentum space when two spin non-degenerate electronic bands (one valence and one conduction) linearly cross or touch each other near the Fermi-level forming a topologically non-trivial gapless point in the energy–momentum space. Depending upon the dispersion of valence and conduction bands, Weyl points can be categorized into two types: (i) Type-I, when Weyl cone is not titled and the crossing bands have opposite Fermi-velocity [43–45], and (ii) type-II, when Weyl cone is titled such that Weyl points appear at the touching point of an electron and a hole pocket. In the later case, the crossing bands have unidirectional Fermi-velocity with different magnitudes. When such band crossing occurs between spin degenerate bands, thus formed gapless points are referred as Dirac points. Analogous to Weyl points, Dirac points can also be classified into two type-I and type-II categories.

We can use the PyProcar code to check the spin degeneracy of electronic bands near the crossing points and determine the type of gapless point by analyzing the dispersion of bands in the vicinity of the crossing point. Fig. 12 shows the spin projected bandstructure of TaSb, which is a topological metal hosting both type-I and type-II Weyl points. For more details regarding the topological properties of TaSb and other similar topological metals, we refer the reader to Refs. [45,46].

3.10. Compare bands

This module is useful to compare different bands from different materials on the same band plot (Fig. 13). The bands are plotted for the same k-path in order to have a meaningful comparison but they do not need to have the same number of k-points in each interval. The `bandscompare()` function contains all the parameters that are used in the `bandsplot()` along with an added feature of displaying a **legend** to help differentiate between the two different band structures. Different **marker** styles can be used as well.

Usage:

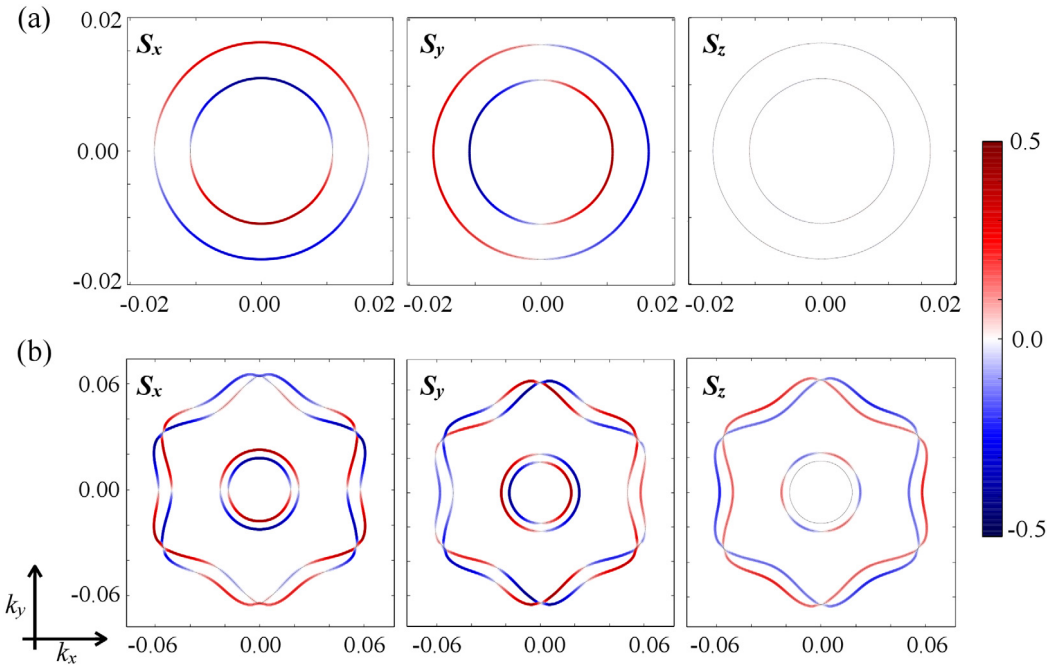

Fig. 10. Spin texture in BiSb monolayer, calculated at a constant energy surface, above and below the Fermi-level, respectively, at (a) $E = E_F + 0.60$ eV, and (b) $E = E_F - 0.90$ eV in a $k_x - k_y$ plane centered at the Γ point [40]. Color depicts the spin projection. 2D spin texture plots reveal the presence of a Rashba type spin-splitting (linear in k) of electronic bands near the Γ point in BiSb monolayer. Warping effects arising due to the k^3 terms in the lower energy valence bands can also be noticed in Fig. 10(b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

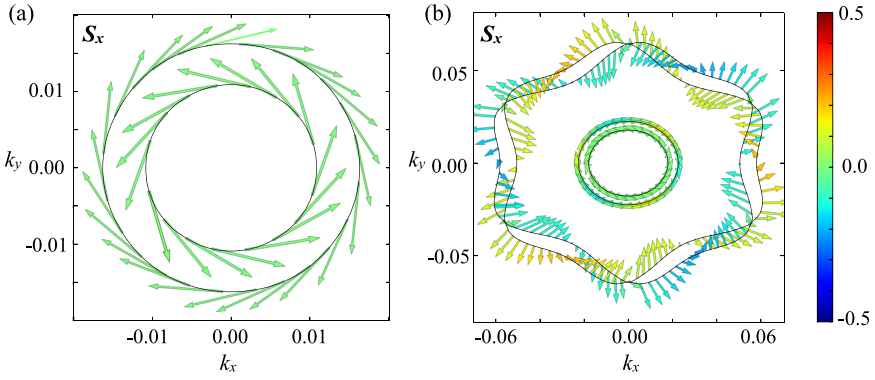

Fig. 11. Projection of S_x spin component shown using arrows instead of heat map as in Fig. 10. Spin texture computed at a constant energy surface (a) $E = E_F + 0.60$ eV, and (b) $E = E_F - 0.90$ eV. All other details are same as in Fig. 10. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

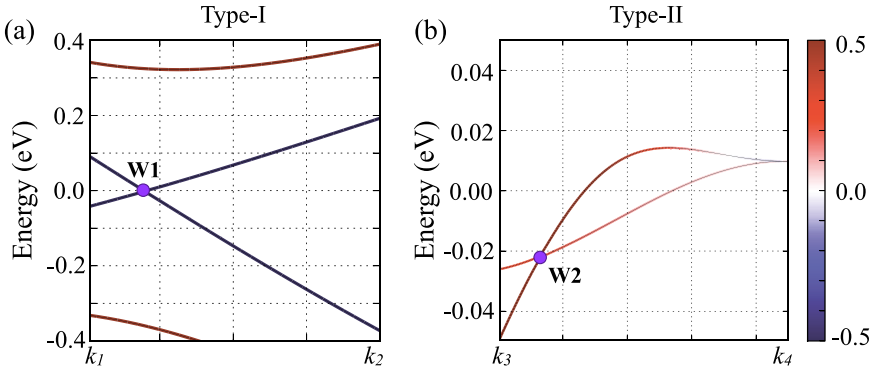

Fig. 12. Projection of one of the spin components (S_x) on the bandstructure of TaSb near the type-I (a) and type-II (b) Weyl points [46]. Spin non-degenerate bands with linear dispersion forming a gapless point near the Fermi-level can be observed. Color depicts the spin projection on electronic bands. Direct coordinates of k_1, k_2, k_3, and k_4 are (0.5, −0.25, 0.0421), (0.430, −0.215, 0.0421), (−0.040, −0.040, 0.357), and (0.000, 0.000, 0.357), respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

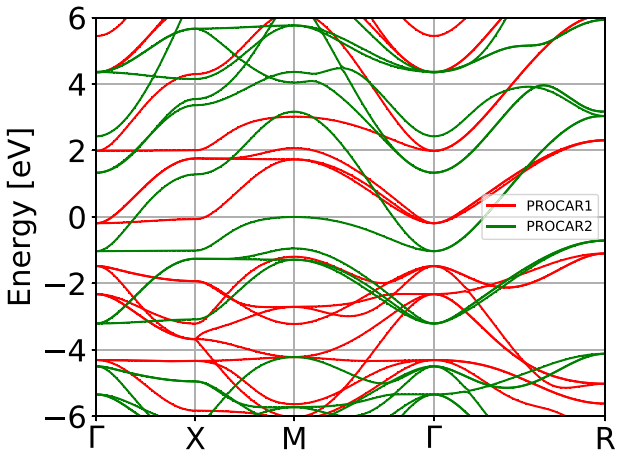

Fig. 13. The comparison of energy bands for two systems of $SrVO_3$ with a shift in their Fermi energy. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

```
pyprocar.bandscompare('PROCAR1','PROCAR2',outcar='OUTCAR1',outcar2='OUTCAR2',cmap='seismic',mode='parametric',marker='*',marker2
    ='-.',elimit=[-5,5],kpointsfile='KPOINTS',legend='PRO1',legend2='PRO2')
```

3.11. Concatenating multiple calculations

Multiple PROCAR files from multiple DFT calculations can be combined with this utility. For instance, performing DFT calculations for MgB_2 supercells are very computationally expensive to do in a single run. This utility is particularly useful in such cases of large systems, where one can split the bandstructure calculations along different high-symmetry directions in BZ, and then concatenate the PROCAR files for each separate k-paths, and finally plot the full bandstructure in a single plot. The following command concatenates the PROCAR files obtained from three separate bandstructure calculations done along Γ-K, K-M, and M-Γ k-path in hexagonal Brillouin zone.

Usage:

```
pyprocar.cat(['PROCAR_G-K','PROCAR_K-M','PROCAR_M-G'],'PROCAR_merged')
```

To concatenate PROCAR's generated from Abinit assuming the files are all in the same directory, use the following command:

Usage:

```
pyprocar.mergeabinit('PROCAR_merged')
```

3.12. Band unfolding

Often times, we need to perform DFT calculations for a supercell geometry rather than the primitive cell. In such cases the band structure becomes quite sophisticated due to the folding of the BZ, and it is difficult to compare the band structure of supercell with that of the primitive cell. The purpose of the band unfolding scheme [47–50] is to represent the bands within the primitive cell BZ. By calculating the unfolding weight function [50] and plotting the fat bands with the line width proportional to the weight, the unfolded bands can be highlighted. Here we use a $2 \times 2 \times 2$ MgB_2 supercell as an example to show the unfolding of band structure. In the bulk structure, where the primitive cell translation symmetry is preserved, the unfolded band is exactly the same as calculated from primitive cell structure. In a structure with a B atom replaced by Al, the translation symmetry can be seen as approximated, and we can still get an unfolded band, with some band smearing out. By comparing these two, we can clearly see the shifting and smearing of the band. The ASE library [51] was employed to perform atomic and cell manipulations required for the unfolding (see Fig. 14).
Usage: First, calculate the band structure in the primitive cell BZ. The PROCAR should be produced with the phase factor included, by setting LORBIT = 12 in VASP. Then the unfold module can be used to plot the unfolded band.

```
import numpy as np
pyprocar.unfold(
        fname='PROCAR',
        poscar='POSCAR',
        outcar='OUTCAR',
        supercell_matrix=np.diag([2, 2, 2]),
        efermi=None,
        shift_efermi=True,
        elimit=(-5, 15),
        kticks=[0, 36, 54, 86, 110, 147, 165, 199],
        knames=['$\Gamma$', 'K', 'M', '$\Gamma$', 'A', 'H', 'L', 'A'],
        print_kpts=False,
        show_band=True,
        savefig='unfolded_band.png')
```

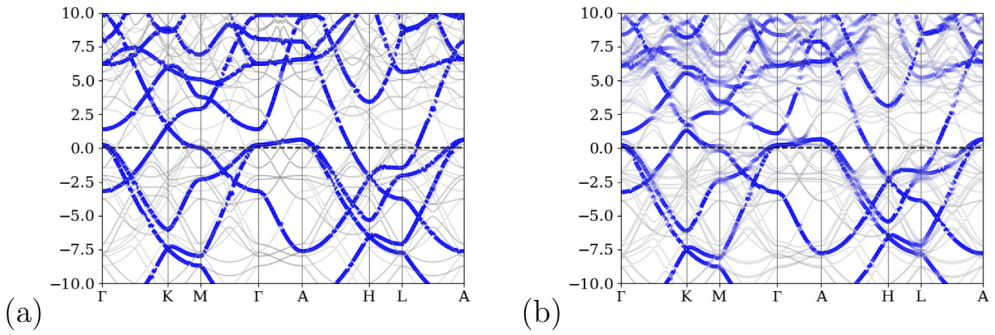

Fig. 14. Band structure unfolded into primitive cell BZ of 2 × 2 × 2 supercell (a) MgB_2 bulk structure, and (b) MgB_2 with one B atom replaced by Al in a 2 × 2 × 2 supercell. The gray lines show the original bands. The width of the blue line denotes the weight of the unfolding. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Conclusion

PyProcar is a user friendly, open source Python library that can be easily used for a variety of DFT pre- and post-processing calculations. We have demonstrated its capability through providing examples for each functionality through a set of four materials with unique characteristics. PyProcar's main specialty lies in its ability to project spin, orbitals and atoms in band structures and 2D and 3D Fermi and constant energy surfaces without involving lengthy, complex syntax. The code is freely available to download at https://github.com/romerogroup/pyprocar, and via `pip`. A standalone version of the library, **procar.py** is located in the github repository. An easy to follow user manual is available at https://romerogroup.github.io/pyprocar/. The PROCAR format is easy to implement in any DFT code, rendering PyProcar accessible across a wide range of DFT codes. We hope this tool will be useful to computational materials scientists in exploring state-of-the-art novel material which would in turn immensely impact the materials community.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work used the XSEDE which is supported by National Science Foundation, USA Grant Number ACI-1053575. XH and EB acknowledge the ARC, USA project AIMED and the F.R.S-FNRS, Belgium PDR project MaRePeThe (GA 19528980). The authors also acknowledge the support from the Texas Advances Computer Center (with the Stampede2 and Bridges supercomputers), the PRACE project TheDeNoMo and on the CECI facilities funded by F.R.S-FNRS, Belgium (Grant No. 2.5020.1) and Tier-1 supercomputer of the Fédération Wallonie-Bruxelles funded by the Walloon Region, Belgium (Grant No. 1117545). This work was supported by the DMREF-National Science Foundation, USA 1434897, National Science Foundation, USA OAC-1740111 and DOE, USA DE-SC0016176 projects. We acknowledge the West Virginia University supercomputing clusters; Spruce Knob and Thorny Flat which were used for the development of the library. FM acknowledges the support from Fondecyt, Chile grants #1150806, #1191353, the Center for the Development of Nanoscience and Nanotechnology CEDENNA FB-0807 and the supercomputing infrastructure of the NLHPC (ECM-02). A special thanks goes to Dr. Guillermo Avendaño Franco for his invaluable support.

References

[1] N.M. Harrison, An Introduction to Density Functional Theory, 26.
[2] W. Kohn, L.J. Sham, Phys. Rev. 140 (1965) A1133–A1138, http://dx.doi.org/10.1103/PhysRev.140.A1133, URL https://link.aps.org/doi/10.1103/PhysRev.140.A1133.
[3] P. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864–B871, http://dx.doi.org/10.1103/PhysRev.136.B864, URL https://link.aps.org/doi/10.1103/PhysRev.136.B864.
[4] X. Gonze, F. Jollet, F.A. Araujo, D. Adams, B. Amadon, T. Applencourt, C. Audouze, J.-M. Beuken, J. Bieder, A. Bokhanchuk, et al., Comput. Phys. Comm. 205 (2016) 106–131.
[5] G. Kresse, J. Hafner, Phys. Rev. B 47 (1993) 558–561, http://dx.doi.org/10.1103/PhysRevB.47.558, URL https://link.aps.org/doi/10.1103/PhysRevB.47.558.
[6] G. Kresse, J. Furthmüller, Comput. Mater. Sci. 6 (1) (1996) 15–50, http://dx.doi.org/10.1016/0927-0256(96)00008-0, URL http://www.sciencedirect.com/science/article/pii/0927025696000080.
[7] J.M. Soler, E. Artacho, J.D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, J. Phys.: Condens. Matter 14 (11) (2002) 2745–2779, http://dx.doi.org/10.1088/0953-8984/14/11/302, URL https://doi.org/10.1088/0953-8984/14/11/302.
[8] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M.B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A.D. Corso, S. de Gironcoli, P. Delugas, R.A. DiStasio Jr., A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E.K. cükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N.L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A.P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, J. Phys.: Condens. Matter 29 (46) (2017) 465901, URL http://stacks.iop.org/0953-8984/29/i=46/a=465901.
[9] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G.L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A.P. Seitsonen, A. Smogunov, P. Umari, R.M. Wentzcovitch, J. Phys.: Condens. Matter 21 (39) (2009) 19, 395502. URL http://www.quantum-espresso.org.
[10] A. Jain, S.P. Ong, G. Hautier, W. Chen, W.D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, K.a. Persson, APL Mater. 1 (1) (2013) 011002, http://dx.doi.org/10.1063/1.4812323, URL http://link.aip.org/link/AMPADS/v1/i1/p011002/s1&Agg=doi.

[11] S. Curtarolo, W. Setyawan, G.L. Hart, M. Jahnatek, R.V. Chepulskii, R.H. Taylor, S. Wang, J. Xue, K. Yang, O. Levy, M.J. Mehl, H.T. Stokes, D.O. Demchenko, D. Morgan, Comput. Mater. Sci. 58 (2012) 218–226, http://dx.doi.org/10.1016/j.commatsci.2012.02.005, URL https://linkinghub.elsevier.com/retrieve/i/S0927025612000717.
[12] T. Zhang, Y. Jiang, Z. Song, H. Huang, Y. He, Z. Fang, H. Weng, C. Fang, Nature 566 (7745) (2019) 475–479, http://dx.doi.org/10.1038/s41586-019-0944-6, URL http://www.nature.com/articles/s41586-019-0944-6.
[13] Editorial, Nat. Phys. 12 (2016) 105, http://dx.doi.org/10.1038/nphys3668, URL https://www.nature.com/articles/nphys3668.
[14] PyProcar - Google groups, URL https://groups.google.com/forum/#!forum/pyprocar.
[15] Google bigQuery, URL https://bigquery.cloud.google.com/results/consummate-web-210605:US.bquijob_75506e9f_16b045620cd?pli=1.
[16] PyPI download stats, URL https://pypistats.org/packages/pyprocar.
[17] PyPI – the Python package index PyPI, URL https://pypi.org/.
[18] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868, http://dx.doi.org/10.1103/PhysRevLett.77.3865, URL https://link.aps.org/doi/10.1103/PhysRevLett.77.3865.
[19] H.J. Monkhorst, J.D. Pack, Phys. Rev. B 13 (1976) 5188–5192, http://dx.doi.org/10.1103/PhysRevB.13.5188, URL https://link.aps.org/doi/10.1103/PhysRevB.13.5188.
[20] G.I. Csonka, J.P. Perdew, A. Ruzsinszky, P.H. Philipsen, S. Lebègue, J. Paier, O.A. Vydrov, J.G. Ángyán, Phys. Rev. B 79 (15) (2009) 155107.
[21] Home, URL https://www.materialscloud.org/home#.
[22] M. Aroyo, J. Perez-Mato, D. Orobengoa, E. Tasci, G. De La Flor, A. Kirov, Bulg. Chem. Commun. 43 (2) (2011) 183–197, cited By 187, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-80955140447&partnerID=40&md5=488772b9e21d2636a3952f66ae80ae84.
[23] M.I. Aroyo, A. Kirov, C. Capillas, J.M. Perez-Mato, H. Wondratschek, Acta Crystallogr. Sect. A 62 (2) (2006) 115–128, http://dx.doi.org/10.1107/S0108767305040286, URL https://doi.org/10.1107/S0108767305040286.
[24] M. Aroyo, J. Perez-Mato, C. Capillas, E. Kroumova, S. Ivantchev, G. Madariaga, A. Kirov, H. Wondratschek, Z. Kristallogr. 221 (2006) 15–27, http://dx.doi.org/10.1524/zkri.2006.221.1.15.
[25] Y. Hinuma, G. Pizzi, Y. Kumagai, F. Oba, I. Tanaka, Comput. Mater. Sci. 128 (2017) 140–184, http://dx.doi.org/10.1016/j.commatsci.2016.10.015, URL http://www.sciencedirect.com/science/article/pii/S0927025616305110.
[26] Python modules documentation — seekpath 1.8.4 documentation, URL https://seekpath.readthedocs.io/en/latest/module_guide/index.html.
[27] J.D. Hunter, Comput. Sci. Eng. 9 (3) (2007) 90–95.
[28] S. van der Walt, J.L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J.D. Warner, N. Yager, E. Gouillart, T. Yu, PeerJ 2 (2014) e453, http://dx.doi.org/10.7717/peerj.453, URL https://doi.org/10.7717/peerj.453.
[29] P. Ramachandran, G. Varoquaux, Comput. Sci. Eng. 13 (2) (2011) 40–51.
[30] J.D. Hunter, Comput. Sci. Eng. 9 (3) (2007) 90–95, http://dx.doi.org/10.1109/MCSE.2007.55.
[31] P.T. Inc., Collaborative Data Science, Plotly Technologies Inc., Montreal, QC, 2015, URL https://plot.ly.
[32] maartenbreddels/ipyvolume: 3d plotting for Python in the Jupyter notebook based on IPython widgets using WebGL, URL https://github.com/maartenbreddels/ipyvolume.
[33] J.A. Silva-Guillén, Y. Noat, T. Cren, W. Sacks, E. Canadell, P. Ordejón, Phys. Rev. B 92 (2015) 064514, http://dx.doi.org/10.1103/PhysRevB.92.064514, URL https://link.aps.org/doi/10.1103/PhysRevB.92.064514.
[34] N. Marzari, A.A. Mostofi, J.R. Yates, I. Souza, D. Vanderbilt, Rev. Modern Phys. 84 (4) (2012) 1419.
[35] S. Singh, C. Espejo, A.H. Romero, Phys. Rev. B 98 (2018) 155309, http://dx.doi.org/10.1103/PhysRevB.98.155309, URL https://link.aps.org/doi/10.1103/PhysRevB.98.155309.
[36] S. Singh, A.M. Alsharari, S.E. Ulloa, A.H. Romero, Proximity-induced topological transition and strain-induced charge transfer in graphene/mos2 bilayer heterostructures, 2018, arXiv preprint arXiv:1806.11469.
[37] S. Singh, W. Ibarra-Hernández, I. Valencia-Jaime, G. Avendaño-Franco, A.H. Romero, Phys. Chem. Chem. Phys. 18 (2016) 29771–29785, http://dx.doi.org/10.1039/C6CP05401C, URL http://dx.doi.org/10.1039/C6CP05401C.
[38] S. Singh, A.C. Garcia-Castro, I. Valencia-Jaime, F. Muñoz, A.H. Romero, Phys. Rev. B 94 (2016) 161116, http://dx.doi.org/10.1103/PhysRevB.94.161116, URL https://link.aps.org/doi/10.1103/PhysRevB.94.161116.
[39] A.C. Garcia-Castro, M.G. Vergniory, E. Bousquet, A.H. Romero, Phys. Rev. B 93 (2016) 045405, http://dx.doi.org/10.1103/PhysRevB.93.045405, URL https://link.aps.org/doi/10.1103/PhysRevB.93.045405.
[40] S. Singh, A.H. Romero, Phys. Rev. B 95 (2017) 165444, http://dx.doi.org/10.1103/PhysRevB.95.165444, URL https://link.aps.org/doi/10.1103/PhysRevB.95.165444.
[41] S.K. Singh, Structural Prediction and Theoretical Characterization of Bi-Sb Binaries: Spin-Orbit Coupling Effects (Ph.D. thesis), West Virginia University, 2018.
[42] S. Singh, Z. Zanolli, M. Amsler, B. Belhadji, J.O. Sofo, M.J. Verstraete, A.H. Romero, Low energy phases of bilayer bi predicted by structure search in two dimensions, 2019, arXiv preprint:1901.05060.
[43] A.A. Soluyanov, D. Gresch, Z. Wang, Q. Wu, M. Troyer, X. Dai, B.A. Bernevig, Nature 527 (2015) 495 EP – URL https://doi.org/10.1038/nature15768.
[44] B. Yan, C. Felser, Ann. Rev. Condens. Matter Phys. 8 (1) (2017) 337–354, http://dx.doi.org/10.1146/annurev-conmatphys-031016-025458, URL https://doi.org/10.1146/annurev-conmatphys-031016-025458.
[45] G.W. Winkler, S. Singh, A.A. Soluyanov, Topology of triple-point metals, 2019, arXiv:1903.01269.
[46] S. Singh, Q. Wu, C. Yue, A.H. Romero, A.A. Soluyanov, Phys. Rev. Mater. 2 (2018) 114204, http://dx.doi.org/10.1103/PhysRevMaterials.2.114204, URL https://link.aps.org/doi/10.1103/PhysRevMaterials.2.114204.
[47] T.B. Boykin, G. Klimeck, Phys. Rev. B 71 (2005) 115215, http://dx.doi.org/10.1103/PhysRevB.71.115215, URL https://link.aps.org/doi/10.1103/PhysRevB.71.115215.
[48] W. Ku, T. Berlijn, C.-C. Lee, Phys. Rev. Lett. 104 (2010) 216401, http://dx.doi.org/10.1103/PhysRevLett.104.216401, URL https://link.aps.org/doi/10.1103/PhysRevLett.104.216401.
[49] V. Popescu, A. Zunger, Phys. Rev. B 85 (2012) 085201, http://dx.doi.org/10.1103/PhysRevB.85.085201, URL https://link.aps.org/doi/10.1103/PhysRevB.85.085201.
[50] P.B. Allen, T. Berlijn, D.A. Casavant, J.M. Soler, Phys. Rev. B 87 (2013) 085322, http://dx.doi.org/10.1103/PhysRevB.87.085322, URL https://link.aps.org/doi/10.1103/PhysRevB.87.085322.
[51] A.H. Larsen, J.J. Mortensen, J. Blomqvist, I.E. Castelli, R. Christensen, M. Dułak, J. Friis, M.N. Groves, B. Hammer, C. Hargus, E.D. Hermes, P.C. Jennings, P.B. Jensen, J. Kermode, J.R. Kitchin, E.L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J.B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K.S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, K.W. Jacobsen, J. Phys.: Condens. Matter 29 (27) (2017) 273002, http://dx.doi.org/10.1088/1361-648x/aa680e, URL https://doi.org/10.1088/1361-648x/aa680e.