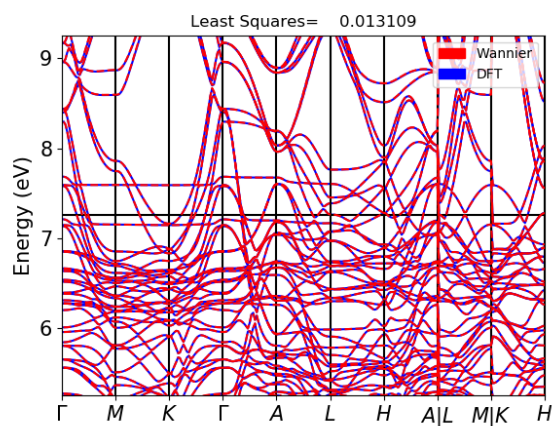


Notes on: Automated Wannier Functions construction

Ilias Samathrakis

February 16, 2022



Contents

1	Basic idea	2
2	Input	4
3	Output	10
4	The code in practice	11
4.1	Template	11
4.2	Code	12
5	Examples	12
6	Appendix	17
6.1	Constrained calculations	17
6.2	LDA+U	17
6.3	Submission file	17

The following colour code is used throughout the notes to make the distinction of items easier.

Colour	Explanation
green	File of the scheme
purple	Output file
red	External Software
blue	Variable in the code
orange	Function in the code

Table 1: Colour explanation throughout the notes

1 Basic idea

The main goal of the automated wannier functions construction scheme is to provide a general framework in order to compute the wannier functions of any 3D system (excluding f-block elements) automatically. This is achieved by combining the DFT package **VASP** [1, 2, 3, 4] and the wannier functions package **Wannier90** [5].

The used methodology of constructing Wannier functions consists of the four calculation steps:

1. Self consistent calculations using **VASP**
2. Density of states calculations using **VASP**
3. Wannier functions construction using **Wannier90**
4. Band structure calculations using **VASP**

More details on the methodology are found in [6]

The framework targets at:

- generating the input files needed for **VASP**
- generating the input files needed for **Wannier90**
- plotting based on output

- identifying the missing parts due to errors in the procedure

The following external files and/or software are required to run the scheme:

- **POSCAR**
- **VASP** (tested in version: 5.2.12)
- **Pseudopotentials for VASP**
- **Wannier90** (tested in version: 2.0.0)
- **Python3**
- **Pymatgen**

The scheme is based on several in-house developed scripts written in **Python3**, which include:

- **vasp_input_hfe_cif.py**
File to generate the input of **VASP**
- **input_parameters.py**
The input file. Calculation parameters are defined here.
- **highsymk.py**
File to generate the input of **VASP**
- **autoconstruction.py**
File to generate the input file of **Wannier90**
- **plotband.py**
File to plot the band structure obtained from **VASP**
- **compareband.py**
File to plot the band structure obtained from **VASP** and **Wannier90** in a single plot.

In the following, the reader can get familiar with the input file and practice the acquired knowledge following the detailed given examples.

2 Input

The input file of the automatic wannier functions construction scheme is called `input_parameters.py`. It consists of six python functions which are listed and explained in more detail below:

1. Function `calc_type()`

```
1 def calc_type():
2     ## Type of Calculations you want to perform.
3
4     SC = True      ## Self consistent
5     DOS = True     ## Density of states
6     WANN = True    ## Wannier functions construction
7     BAND = True    ## Band structure
8
9     SMART_SEARCH = False ## Searches and determines
    the progress of each directory. If set 'True', none of
    the previous tags is valid.
10
11     CALCS = [SC,DOS,WANN,BAND]
12
13     return CALCS, SMART_SEARCH
```

The variables of the `calc_type()` function and their meaning is summarised in Tab 2.

Variable	Type	Explanation
SC	boolean	Self consistent
DOS	boolean	Density of states
WANN	boolean	Wannier functions
BAND	boolean	Band structure
SMART_SEARCH	boolean	Switch between the two functionalities

Table 2: Variables in `calc_type()`

The code provides two different functionalities which are based on the choice of the tag `SMART_SEARCH`.

(a) `SMART_SEARCH = FALSE`

The code generates the input files of each of the steps their variables defined as TRUE.

(b) `SMART_SEARCH = TRUE`

The code ignores the rest variables and determines the progress of the calculation by reading the output files of `VASP` and `wannier90`. This functionality is used to automatically determine the missing steps of the procedure and only resubmit these parts.

2. Function `running_parameters()`

```
1 def running_parameters():
2     cores = 24
3     time = 24      ## in hours (always integer)
4     memory = 0     ## in GB (if 0, it is set
5     automatically)
6     pr_id = ""     ## Project id (you can find it
7     using csreport command)
8
9     return cores, time, memory
```

The variables of the `running_parameters()` function and their meaning is summarised in Tab. 3.

Variable	Type	Explanation
cores	$n, n \in \mathbb{Z}^+$	number of cores
time	$n, n \in \mathbb{Z}^+$	time in hours
memory	$n, n \in \mathbb{Z}$	memory in GB
pr_id	string	project id

Table 3: Variables in `running_parameters()`

The code provides two different ways of defining the variable `memory`

(a) `memory \neq 0`

The code sets the memory based on the provided value (integer only)

(b) `memory = 0`

The code automatically determines the memory based on experience acquired from previous calculations (preferable for high throughput).

3. Function `directories()`

```
1 def directories():
2     vasp_dir = ''    ## Path of VASP's executable
3     wannier90_dir = '' ## Path of Wannier90's
4     executable
5     pseudopotentials_dir = '' ## Path of
6     pseudopotentials
7     python_dir = '' ## Path of python#
8
9     vasp_output = '' ## Output file of VASP (only
10    name, no path) eg. vasp.log
11    wannier_output = '' ## Output file of wannier90
12    (only name, no path) eg. wannier90.wout
13
14    return vasp_dir, wannier90_dir,
15           pseudopotentials_dir, vasp_output, wannier_output
```

The variables of the `directories()` function and their meaning is summarised in Tab. 4.

Variable	Type	Explanation
vasp_dir	string	Path of VASP executable
wannier90_dir	string	Path of wannier90 executable
pseudopotentials_dir	string	Path of Pseudopotentials
python_dir	string	Path of python
vasp_output	string	Name of VASP output file
wannier_output	string	Name of wannier90 output file

Table 4: Variables in `directories()`

4. Function `hte_tags()`

```
1 def hte_tags():
2     ## Input for INCAR and KPOINTS. Modify
3     accordingly
4
5     magnetism = True    ## True if the system
6     is magnetic, False otherwise
7     non_collinearity = True ## True if non-
8     collinear system, False otherwise. It is set to False
9     if magnetism is False
```

```

6         SOC                      = True    ## True if you want to
        include SOC, False otherwise. It is set to False if
        magnetism is False
7         constraint                = False  ## True if you want to
        constrain along a certain direction. Set MAGMOM to
        specify the direction
8         U_calc                   = True    ## True if you want L(S
        )DA+U calculations, False otherwise
9
10        ktype = "M"                ## M for Monkhorst Pack
        , G for Gamma
11
12        kspace_density            = 50     ## KPOINTS
13        kspace_density_wfs        = 40     ## KPOINTS.wfs
14        kspace_density_dos        = 60     ## KPOINTS.dos
15
16        ## NPAR is set automatically
17        ## MAGMOM is parallel to x axis by default. Set
        it appropriately for different directions
18        ## NBANDS is calculated automatically if it is
        not set (preferable for high throughput)
19        ## ICHARG, LWANNIER90, are set automatically
20
21        incar_tags = {
22
23            "prec" : "Accurate",
24            "algo" : "NORMAL",
25            "encut" : 500,
26            "lorbit" : 10,
27            "ediff" : 0.000001,
28            "ediffg" : -0.001,
29            "amix" : 0.001,
30            "bmix" : 0.0001,
31            "isym" : -1,
32            "nelm" : 500,
33            "istart" : 1,
34            "ismear" : 1,
35            "sigma" : 0.06,
36        }
37
38        return incar_tags, magnetism, non_collinearity,
        SOC, U_calc, kspace_density, kspace_density_wfs,
        kspace_density_dos

```

The variables of the `hte_tags()` function and their meaning is sum-

marised in Tab. 5

Variable	Type	Explanation
magnetism	boolean	Switches magnetism on and off
non_collinearity	boolean	Switches non_collinearity on and off
SOC	boolean	Switches SOC on and off
constraint	boolean	Switches constraint on and off
U_calc	boolean	Switches L(S)DA+U on and off
ktype	M or G	M for Monkhorst Pack, G for Gamma grid
kspace_density	$n, n \in \mathbb{Z}^+$	kspace density of the SC step
kspace_density_wfs	$n, n \in \mathbb{Z}^+$	kspace density of the WFS step
kspace_density_dos	$n, n \in \mathbb{Z}^+$	kspace density of the DOS step
incar_tags		See VASP documentation

Table 5: Variables in `directories()`

The dictionary `incar_tags` may contain all **VASP** tags available here
However, certain limitations apply:

- Tag NPAR is computed automatically. Do not set.
- Tag MAGMOM is set as parallel to x-axis for every atom by default. If another magnetisation direction is the desired, please set it appropriately. Consider the length of the string based on the tags `NON_COLLINEARITY` and `SOC`.
- Tag NBANDS is set automatically if not defined. It is advised not to set it unless a specific reason exists.
- If `constraint = True`, MAGMOM should exist since it defines the constrained direction. Additionally, its length has to be consistent with the values of `NON_COLLINEARITY` and `SOC`. See section 6.1 of the Appendix for more details regarding constrained calculations.

5. Function `pseudopotentials()`

```

1 def pseudopotentials():
2     pseudop = {
3         "H" : "H", "He" : "He", ... , "Og" : "NA"
```



```

4     }
5     return pseudep

```

The pseudopotential of each atom in the `pseudep` dictionary is defined according to Tab. 6.

Input of dictionary <code>pseudep</code>		
Key \rightarrow element	Separator	Value \rightarrow chosen pseudopotential
"H"	:	"H"
"Sn"	:	"Sn_d"

Table 6: Input of `pseudep` dictionary

6. Function `ldau_values()`

This function is active only if `U_calc = TRUE` in `hte_tags()`

```

1 def ldau_values():
2     u_val = {
3         "La" : 9, "Ce" : 9, ... , "Lr" : 9
4     }
5
6     j_val = {
7         "La" : 0.8, "Ce" : 0.8, ... , "Lr" : 0.8
8     }
9
10    l_val = {
11        "La" : 2, "Ce" : 2, ... , "Lr" : 2
12    }
13    return u_val, j_val, l_val

```

The dictionaries `u_val`, `j_val` and `l_val` that correspond to the LDAU, LDAJ and LDAL tags in `VASP` for each chemical element are defined according to Tab. 7.

Input of dictionary u_val		
Key → element “Sm”	Separator :	Value → LDAUU 9
Input of dictionary j_val		
Key → element “Sm”	Separator :	Value → LDAUJ 0.8
Input of dictionary l_val		
Key → element “Sm”	Separator :	Value → LDAUL 2

Table 7: Input of `u_val`, `j_val` and `l_val` dictionaries

The code sets by default `LDAUTYPE = 1` and `LDAUPRINT = 2`. Modification of these **VASP** tags is not possible from the input file but only from the source code file `vasp_input_hte_cif.py`. For more details see the Appendix in Section 6.2.

3 Output

The output files of the scheme depend on the variables set in the function `calc_type` of the `input_parameters.py` file. Tab. 8 summarises the output files obtained by running `vasp_input_hte_cif.py` and Tab. 9 the output of each of the rest files of the scheme.

Variable	Output files
Independent of variables	<code>run.sh</code>
If <code>SC = True</code>	<code>INCAR</code> , <code>KPOINTS</code> , <code>POTCAR</code>
If <code>DOS = True</code>	<code>INCAR.dos</code> , <code>KPOINTS.dos</code> , <code>POTCAR</code>
If <code>WANN = True</code>	<code>INCAR.wfs</code> , <code>KPOINTS.wfs</code> , <code>POTCAR</code>
If <code>BAND = True</code>	<code>INCAR.band</code> , <code>POTCAR</code>

Table 8: Output files of `vasp_input_hte_cif.py`

File of scheme	Output files
autoconstruction.py	wannier90.win
highsymk.py	KPOINTS.band
plotband.py	1.dat, ..., n.dat, band.png
compareband.py	wannband.png
input_parameters.py	No output

Table 9: Output files of the rest files of the scheme

All **INCAR**, all **KPOINTS** and **POTCAR** as well as the **POSCAR** are the input files of **VASP**. **run.sh** is the submission file (see section 6.3 of the Appendix for more details). **wannier90.win** is the input file of **Wannier90**, **1.dat**, ..., **n.dat** are supporting files needed for getting **band.png** which is the band structure along the chosen high symmetric kpath obtained using **VASP**. Finally, **wannband.png** displays the band structure from **VASP** and **Wannier90** in a single plot.

4 The code in practice

All the required files of the scheme are located within the source directory. A bash script “temp-wannflow.sh” as well as the folder “Examples” are included too. The suggested way to use the scheme is to create a template for easy use. Instructions to create the template (once) and to use the code is found in the step-by-step guide below.

4.1 Template

1. Navigate to your home directory, create a directory (template_wann) and paste the content of folder “source” within the newly created directory.

```

1  $ cd
2  $ mkdir template_wann
3  $ cp -r ‘‘directory_of_source’’/source/* template_wann

```

2. Set the path of the created directory in the file “temp-wannflow.sh”
3. Paste “temp-wannflow.sh” in a convenient and easy to access directory (home)

```

1 $ cd
2 $ cp 'directory_of_temp-wannflow.sh' /temp-wannflow.sh
   .

```

4.2 Code

1. Create a directory and navigate within

```

1 $ mkdir my_dir
2 $ cd my_dir

```

2. Paste the “temp-wannflow.sh” file within the directory

```

1 $ cp 'directory_of_POSCAR_file' /POSCAR .
2 $ cp ~/temp-wannflow.sh .

```

3. Run the script

```

1 $ sh temp-wannflow.sh

```

4. Modify “input_parameters.py”

5. Run the scheme

```

1 $ python vasp_input_hte_cif.py

```

5 Examples

1. **Self consistent calculation of MnNi₂Ga**

Directory Example_1 contains the **POSCAR** file of MnNi₂Ga as well as the six files of the scheme.

Task: Create the input files of the self consistent calculation.

Solution:

- (a) Set the input tags in `calc_type()` function of `input_parameters.py` as shown:

```

1 SC = True
2 DOS = False
3 WANN = False
4 BAND = False
5 SMART_SEARCH = False

```

(b) Run the code

```

1 $ python3 vasp_input_hfe_cif.py

```

Output: The files **INCAR**, **POTCAR**, **KPOINTS** and **run.sh** have been created.

2. Density of States calculation of MnNi₂Ga

Directory Example_2 contains the **POSCAR** file of MnNi₂Ga as well as the six files of the scheme.

Task: Create the input files of the self consistent and density of states calculations.

Solution:

(a) Set the input tags in **calc_type()** function of **input_parameters.py** as shown:

```

1 SC = True
2 DOS = True
3 WANN = False
4 BAND = False
5 SMART_SEARCH = False

```

(b) Run the code

```

1 $ python3 vasp_input_hfe_cif.py

```

Output: The files **INCAR**, **INCAR.dos**, **POTCAR**, **KPOINTS**, **KPOINTS.dos** and **run.sh** have been created.

3. Input of all calculations

Directory Example_3 contains the **POSCAR** file of MnNi₂Ga as well as the six files of the scheme.

Task: Create the input files of the self consistent, density of states, wannier functions construction and band structure calculations.

Solution:

- (a) Set the input tags in `calc_type()` function of `input_parameters.py` as shown:

```
1 SC = True
2 DOS = True
3 WANN = True
4 BAND = True
5 SMART_SEARCH = False
```

- (b) Run the code

```
1 $ python3 vasp_input_hfe_cif.py
```

Output: The files `INCAR`, `INCAR.dos`, `INCAR.wfs`, `INCAR.band`, `POTCAR`, `KPOINTS`, `KPOINTS.dos`, `KPOINTS.wfs` and `run.sh` have been created¹.

4. **Wannier90 input of YCo₂**

Directory `Example_4` contains, among others, the `POSCAR` file of YCo₂, `dos` and `wann` directories which correspond to the density of states and the wannier functions construction respectively.

Task: Create the input file of wannier90 calculation.

Solution:

Navigate to the `wann` directory and run `autoconstruction.py` (`dos/vasprun.xml` file is needed).

```
1 $ cd wann/
2 $ python3 autoconstruction.py
```

Output: The file `wannier90.win` has been created.

5. **Band structure calculation of YCo₂**

Directory `Example_5` contains, among others, the `POSCAR` file of YCo₂,

¹KPOINTS.band file is created with `highsymk.py`, see `Example_5`

dos and wann and band directories which correspond to the density of states, the wannier functions construction and the band structure calculations respectively.

Task: Create KPOINTS used for the band structure calculations (KPOINTS.band).

Solution:

Navigate to the band directory and run highsymbk.py

```
1 $ cd band/  
2 $ python3 highsymbk.py
```

Output: The file **KPOINTS** has been created.

6. **SMART_SEARCH** tag

Directory Example_6 contains, among others, the **POSCAR** file of YCo₂, dos and wann directories which correspond to the density of states and the wannier functions construction respectively.

Task: All the necessary directories are present, however, something went wrong during the calculation and some parts are incomplete. Check the output files of each step to find out which part(s) is (are) incomplete. After finding the missing parts, use the code to resubmit only those. (Hint: Are all the directories present before and after running the code?)

Solution:

- (a) Set the input tags in `calc_type()` function of `input_parameters.py` as shown:

```
1 SC = True  
2 DOS = True  
3 WANN = True  
4 BAND = True  
5 SMART_SEARCH = True
```

- (b) Run the code
(**vasp.out**, dos/**vasp.out**, wann/**vasp.out**, band/**vasp.out**
and wann/**wannier90.wout** are needed).

```
1 $ python3 vasp_input_hfe_cif.py
```

Output: Check the `run.sh` file to find the missing parts that are ready to submit.

7. Band structure plot of YCo_2

Directory Example_7 contains, among others, the `POSCAR` file of YCo_2 , `dos` and `wann` and `band` directories which correspond to the density of states, the wannier functions construction and the band structure calculations respectively.

Task: Visualise the band structure obtained from VASP

Solution:

- (a) Navigate to the band directory and run `plotband.py` (band/`PROCAR` file is needed).

```
1 $ cd band/  
2 $ python3 plotband.py
```

- (b) The file `band.png` has been created. Open it using

```
1 $ display band.png
```

8. Band structure plot

Directory Example_8 contains, among others, the `POSCAR` file of YCo_2 , `dos` and `wann` and `band` directories which correspond to the density of states, the wannier functions construction and the band structure calculations respectively.

Task: Visualise the band structure obtained from `VASP` and the one obtained from `Wannier90` in a single plot

Solution:

- (a) Run `compareband.py` (band/`1.dat`,...,`n.dat`, band/`PROCAR`, wann/`wannier90_hr.dat` files are needed).


```
1 $ python3 compareband.py
```

(b) The file wannband.png has been created. Open it using

```
1 $ display wannband.png
```

6 Appendix

6.1 Constrained calculations

Modification of **VASP** tags `I_CONSTRAINED_M` and `LAMBDA` is only possible from the source code file `vasp_input_hfe_cif.py`. Specifically, in function `get_additional_tags()` they are defined in the form:

```
1 incar_tags.append(['i_constrained_m'.upper(),1])
2 incar_tags.append(['lambda'.upper(),10])
```

6.2 LDA+U

Modification of **VASP** tags `LDAUTYPE` and `LDAUPRINT` is only possible from the source code file `vasp_input_hfe_cif.py`. Specifically, in function `get_additional_tags()` they are defined in the form:

```
1 incar_tags.append(['ldautype'.upper(),1])
2 incar_tags.append(['ldauprint'.upper(),2])
```

6.3 Submission file

In case the calculations are not performed in the Lichtenberg High Performance Computer of TU Darmstadt (HRZ), please modify the generation of the submission file in the function `get_run()` (from line 345) of the source code file `vasp_input_hfe_cif.py`

References

- [1] G. Kresse and J. Hafner, “Ab initio molecular dynamics for liquid metals,” *Phys. Rev. B*, vol. 47, pp. 558–561, Jan 1993.

- [2] G. Kresse and J. Hafner, “Ab initio molecular-dynamics simulation of the liquid-metal–amorphous-semiconductor transition in germanium,” Phys. Rev. B, vol. 49, pp. 14251–14269, May 1994.
- [3] G. Kresse and J. Furthmüller, “Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set,” Computational Materials Science, vol. 6, no. 1, pp. 15 – 50, 1996.
- [4] G. Kresse and J. Furthmüller, “Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set,” Phys. Rev. B, vol. 54, pp. 11169–11186, Oct 1996.
- [5] A. A. Mostofi, J. R. Yates, G. Pizzi, Y.-S. Lee, I. Souza, D. Vanderbilt, and N. Marzari, “An updated version of wannier90: A tool for obtaining maximally-localised wannier functions,” Computer Physics Communications, vol. 185, no. 8, pp. 2309 – 2310, 2014.
- [6] Z. Zhang, R.-W. Zhang, X. Li, K. Koepernik, Y. Yao, and H. Zhang, “High-throughput screening and automated processing toward novel topological insulators,” The journal of physical chemistry letters, vol. 9, no. 21, pp. 6224–6231, 2018.