Wannier90 calculation practice

Pengfei Suo

November 9, 2021

1 Abstract

Wannier tight-binding models are effective models constructed from first-principles calculations. As such, they bridge a gap between the accuracy of first-principles calculations and the computational simplicity of effective models. Maximally localized Wannier functions (MLWFs)[1] has been implemented in the Wannier90[2] software package. Here we show how to construct wannier function of graphite and graphene, combined with three first-principle software package, VASP[3], Quantum-Espresso[4], and GPAW[5].

2 Electronic structure of graphite and graphene

Graphene is the only one atomic layer structure of graphite. They have similar structure and the main difference between them is the dimension. Their layered structure results in $sp2$ hybridization of C atoms. In one graphene layer, a unit cell has 2 C atoms, with 4 valence electrons per atom. This makes π and π^* bond, which is attributed to p_z orbital of the 2 C atoms in the unit cell, contributes the band structure arround fermi level.

Therefore, we have around 3 choices of projection atomic orbitals to construct the wannier functions:

- p_z orbital of 2 C atoms.
- p_z and $sp2$ orbitals of 2 C atoms.
- p_z orbital of 2 C atoms and 3 s-like orbitals from the 3 σ-bonds (rather than anti-bonds) of $sp2$ hybridization of 2 C atoms.

The first one is the simplest choice and many toy-models are constructed by this method. The third one describes occupied states quite well and can solve more problems. The second one is also good, but usually difficult to practice, because more and higher unoccupied states are considered.

In the following, we will show the results from the first and third choices. Before the wannier calculation, we need to have done the high precision (e.g. `EDIFF` = 1E-8) self-consistent-field calculation and obtained the self-consistent density.

3 VASP + wannier90

The interface between wannier90 and QE, GPAW is natural, but for VASP, we have to recompile the code with wannier90 library.

3.1 Configuration

In our test, VASP 6.x version, especially from 6.2.0, can interface with wannier90 v.3.x. However, when writting wannier90.mmn file, it's very very slow. Fortunately, VASP 5.4.4 works very well. Generally speaking, only wannier90 v.1.2 matches VASP 5.x well, but with limited funciton, as shown in Table 1. Here we utilies an interface written by Dr. Chengcheng Xiao (`https://github.com/Chengcheng-Xiao/VASP2WAN90_v2_fix`). Firstly,

```
patch -p0 < mlwf.patch
```

Then, add the following two rows in the VASP `makefile.include`:

```
CPP_OPTIONS += -DVASP2WANNIER90v2
LLIBS += /path/to/your/wannier90-2.1/libwannier.a
```

Finally, we can compile the VASP code using `make all` command.

Table 1: Comparison of wannier90 1.2 vs 2.1

	wannier90 1.2	wannier90 2.1 (fixed)
nosoc	Yes	Yes
nosoc + ferromagnetic	No	Yes
nosoc + antiferromagnetic	No	Yes
nosoc	Yes	Yes
nosoc + ferromagnetic	Yes	Yes
nosoc + antiferromagnetic	Yes	Yes

3.2 General steps

Generally, we need the following 5 steps:

- Do nscf calculation with `wannier90` by setting `LWANNIER90=T` in the `INCAR` file.
  In this step, no `NPAR`, `KPAR` or `NCORE` is allowed in `INCAR`. we also need to set `NBANDS`, and this has to be integral multiple of cores in calculation. After running VASP, we will get `win`, `mmn`, and `eig` files, corresponding to wannier input file, the overlaps of Bloch states at neighboring k-points $\langle u_{n\mathbf{k}}|u_{n\mathbf{k}+\mathbf{b}}\rangle$ file, and Kohn-Sham eigenvalues file, respectively.
- Add `num_bands`, `num_wann`, `exclude_bands` and `projections` information into the `win` file. For wannier90 2.1, we have to delete the `use_bloch_phase` tag (different from 1.2 version).
- Rerun the VASP code to generate the `amn` file, which contains the initial projector overlaps $\langle u_{n\mathbf{k}}|\widetilde{p}_i^a\rangle$ information.
- Add the disentanglement information (`dis_win_max` and `dis_num_iter` tags), band plot information and other useful information (`guiding_centres`, `write_xyz` and `write_hr` tags) into the `win` file.
- Run the `wannier90` code to get the disentangled results.

3.3 Graphene

The `INCAR` file:

```
INCAR created by Atomic Simulation Environment
 ENCUT = 520.000000
 SIGMA = 0.100000
 EDIFF = 1.00e-08
 PREC = Accurate
 ICHARG = 11
 ISMEAR = 0
 ISTART = 1
 LCHARG = F
 LWAVE = F
 LWANNIER90 = T
 NBANDS = 20
```

The projection information for p_z case:

```
num_bands = 19
exclude_bands:1
num_wann = 2

dis_win_max = 9
begin projections
C:pz
end projections
```

while the projection information for $p_z + \sigma$ case:

```
num_bands = 20
num_wann = 5

dis_win_max = 9
begin projections
C:pz
f=0.3333333, 0.1666667, 0.5 : s
f=0.8333333, 0.6666667, 0.5 : s
f=0.8333333, 0.1666667, 0.5 : s
end projections
```

the plot information in `win` file:

```
bands_plot = true
bands_num_points = 100
bands_plot_format = gnuplot

begin kpoint_path
G 0.00000 0.00000 0.00000 K 0.33333 0.33333 0.00000
K 0.33333 0.33333 0.00000 M 0.00000 0.50000 0.00000
M 0.00000 0.50000 0.00000 G 0.00000 0.00000 0.00000
end kpoint_path
```

After running the `wannier90` code, we use the `VASP_band.py` script to get bandstructure from first-principle calculation:

```
from ase.calculators.vasp import Vasp
from ase.io import read
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.use('Agg')
nbands = 20
npoints = int([line for line in open('wannier90_band.kpt')][0])
def calc_scf():
    calc = Vasp(
            command = 'mpirun -n %d vasp_std' % nbands,
            xc = 'LDA',
            setups='recommended',
            kpts=(15,15,1),
            istart=0,
            icharg=2,
            encut=520,
            ncore=4,
            ismear=0,
            sigma=0.1,
            prec='Accurate',
            ediff=1e-6)
    return calc

## bandstructure calculation
def band_calc(atoms):
    calc = calc_scf()
    atoms.calc = calc
    calc.set(
            directory='band')
    atoms.get_potential_energy()
    efermi = float([line for line in open('band/DOSCAR') if line.strip()][5].split()[-2])
    with open('FERMI_ENERGY','w') as f:
        print('# Fermi level in scf calculation',file=f)
```

```
        print('%10.6f' % efermi,file=f)
    calc.set(isym=0,
            kpts={'path':'GKMG','npoints':npoints},
            istart=1,
            icharg=11,
            ismear=0,
            sigma=0.1,
            lorbit=10,
            nbands=nbands,
            lwave=False,
            lcharg=False)
    atoms.get_potential_energy()
    e_nk = calc.band_structure().energies[0].T - efermi     # get band data with reference to efermi
    path = atoms.cell.bandpath('GKMG',npoints=npoints)
    x, X, _ = path.get_linear_kpoint_axis()
    np.savetxt('e_nk.dat',e_nk)           # save the band data
    with open('kpath.dat','w') as f:
        for k in x:
            print(k,file=f)          # save the kpath axis data
    with open('highk.dat','w') as f:
        for k in X:
            print(k,file=f)         # save the high K data

## plot bandstructure
def plot_band(figsize=(6,5)):
    plt.figure(figsize=figsize)
    e_nk = np.loadtxt('e_nk.dat')
    x = np.loadtxt('kpath.dat')
    X = np.loadtxt('highk.dat')
    for e_n in e_nk:
        plt.plot(x, e_n, c='r', lw=2)
    plt.axhline(y=0,c='k',alpha=0.5,lw=1,ls='--')
    for i in X:
        plt.axvline(x=i,c='k',alpha=0.5,lw=1,ls='--')
    plt.axis([x[0],x[-1],-5,5])
    plt.xticks(X,[r'$\Gamma$','K','M',r'$\Gamma$'],size=15)
    plt.yticks(size=14)
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{F}$ (eV)', size=20)
    plt.title('band structure of graphene',size=20)
    plt.savefig('band/band_graphene.png',dpi=600)
    plt.close()

if __name__=='__main__':
    atoms = read('../POSCAR')
    band_calc(atoms)
    plot_band()
```

and plot both the TB band and first-principle band in one figure via `w90_vs_VASP.py` script:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.use('Agg')

seed = 'wannier90'
nwan = int([line for line in open(seed+'.amn')][1].split()[-1])
lw = 1.2
fontsize = 15
```

```
title = 'w90 vs DFT'
ef = np.loadtxt('FERMI_ENERGY')
kx = np.loadtxt('kpath.dat')
X = np.loadtxt('highk.dat')
klabels = [r'$\Gamma$','K','M',r'$\Gamma$']
e_nk = np.loadtxt('e_nk.dat')

## read band data from _band.dat file
ewan_kn=np.zeros((len(kx),nwan),dtype=float)
f=open(seed+'_band.dat')
for i in range(nwan):
    for j in range(len(kx)):
        l=f.readline()
        ewan_kn[j,i]=float(l.split()[1]) - ef
    l=f.readline()
f.close()
ewan_nk = ewan_kn.T

## plot the bandstructrue
def plot_band(window=False):
    for e_k in e_nk:
        plt.plot(kx,e_k,c='r',lw=1.2)
    plt.plot(-1,-1,c='r',lw=1.2,label='DFT')
    for ewan_k in ewan_nk:
        plt.plot(kx,ewan_k,c='b',lw=1.2,ls='--')
    plt.plot(-1,-1,c='b',lw=1.2,ls='--',label='w90')
    for k in X[1:-1]:
        plt.axvline(x=k,c='k',lw=0.5,ls='--')
    plt.axhline(y=0,c='k',lw=0.5,ls='--')
    plt.xticks(X,klabels,size=15)
    plt.yticks(size=10)
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV)',size=15)
    plt.title(title, size=18)
    plt.xlim([0,kx[-1]])
    plt.legend(fontsize=12)
    if window:
        plt.axis([0,kx[-1],-5,5])

plt.figure(figsize=(9,4))
plt.subplot(121)
plot_band()
plt.subplot(122)
plot_band(True)
plt.tight_layout()
plt.savefig('VASP_vs_w90.png',dpi=600)
plt.close()
```

The plotted band structures of p_z and $p_z + \sigma$ configuration are shown in Figure. 1 and 2, respectively.

3.4 Graphite

The `INCAR` file is the same as graphene case, while `KPOINTS` becomse $(15, 15, 3)$. Now there are 4 C atoms in primitive cell, so the corresponding `num_wann` becomes 4. In addition, the `kpoint_path` becomes:

```
begin kpoint_path
G 0.00000 0.00000 0.00000 K 0.33333 0.33333 0.00000
K 0.33333 0.33333 0.00000 M 0.00000 0.50000 0.00000
M 0.00000 0.50000 0.00000 G 0.00000 0.00000 0.00000
G 0.00000 0.00000 0.00000 A 0.00000 0.00000 0.50000
```

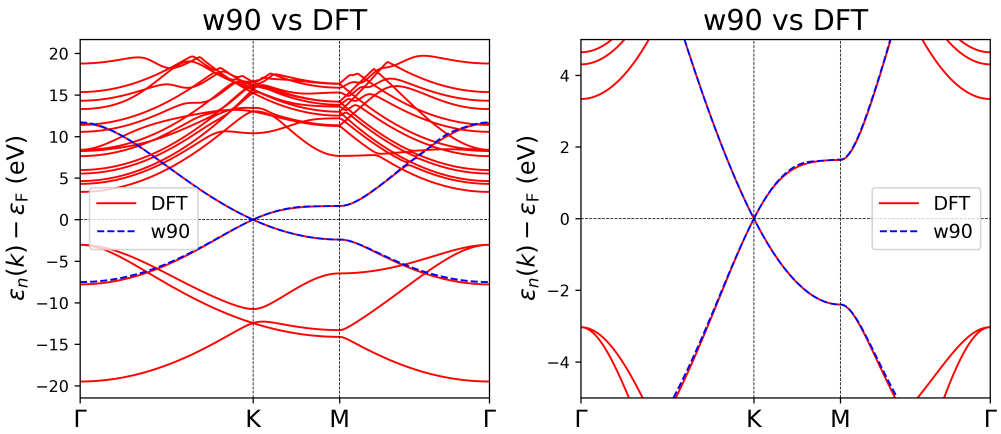

Figure 1: VASP band structure vs wannier p_z interpolated band structure of graphene

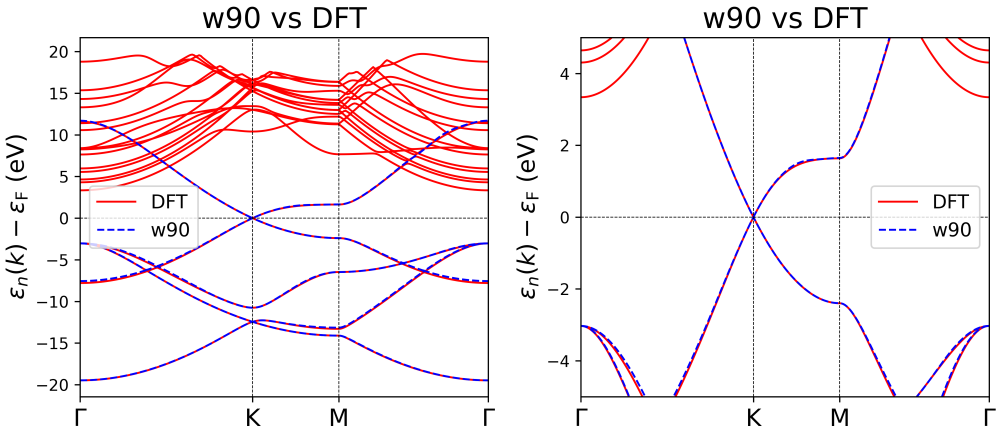

Figure 2: VASP band structure vs wannier $p_z + \sigma$ interpolated band structure of graphene

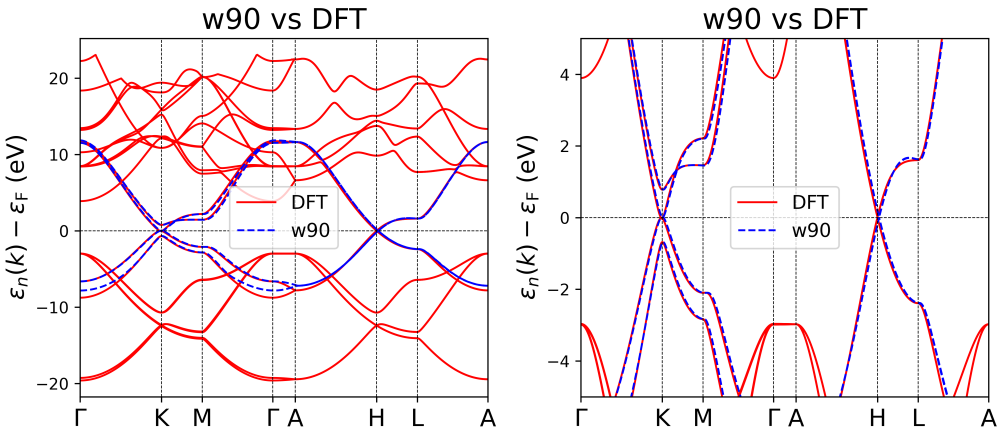

Figure 3: VASP band structure vs wannier p_z interpolated band structure of graphite

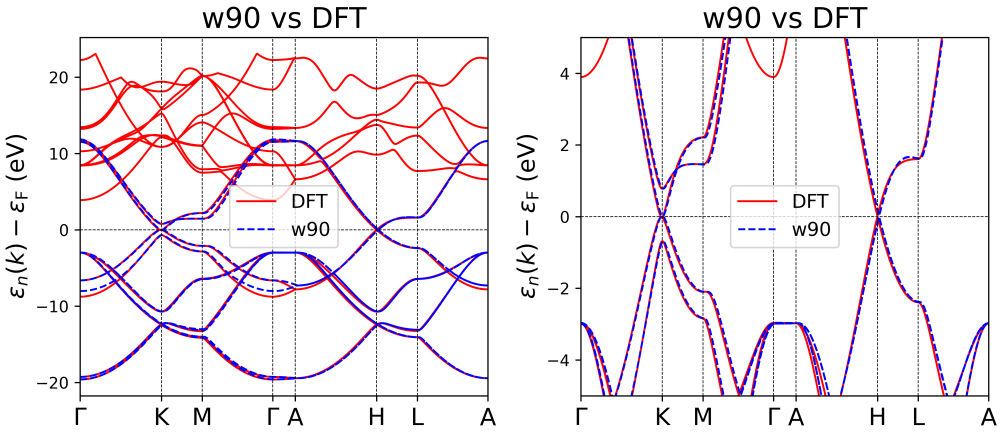

Figure 4: VASP band structure vs wannier $p_z + \sigma$ interpolated band structure of graphite

```
A 0.00000 0.00000 0.50000 H 0.33333 0.33333 0.50000
H 0.33333 0.33333 0.50000 L 0.00000 0.50000 0.50000
L 0.00000 0.50000 0.50000 A 0.00000 0.00000 0.50000
end kpoint_path
```

while the projection information for $p_z + \sigma$ case:

```
num_bands = 20
num_wann = 10

dis_win_max = 17
begin projections
C:pz
f=0.1666667, 0.3333333, 0.0 : s
f=0.6666667, 0.8333333, 0.0 : s
f=0.1666667, 0.8333333, 0.0 : s
f=0.0, 0.5, 0.5 : s
f=0.5, 0.5, 0.5 : s
f=0.5, 1.0, 0.5 : s
end projections
```

The plotted band structures of p_z and $p_z + \sigma$ configuration are shown in Figure. 3 and 4, respectively.

4 QE + wannier90

4.1 General steps

Generally, we need the following 4 steps after scf calculation:

- do nscf calculation with `nosym` and `nbnd`.
  In this step, kpoints are suggested to generate by `kmesh.pl` in `wannier90/utility` directory.
- edit the `win` file and use command `wannier90 -pp filename.win` to get the `nnkp` file.
- edite the `pw2wan.inp` file and utilize `pw2wannier90.x` to generate the `amn`, `mmn`, and `eig` file.
- do `wannier90.x` calculation to construct the wannier functions and plot the results.

In step 2, we usually generate a templete `win` file via a python script `qe2win.py`:

```
import numpy as np
from ase.io import read
import sys

seedname = sys.argv[1]
num_bands = 16
num_wann = 4
dis_num_iter = 1e4
dis_win_max = 21.5
kpts = [15,15,3]
kpath=np.array([[0.,0.,0.],
                [1/3,1/3,0.],
                [.5,0.,0.],
                [0.,0.,0.],
                [0.,0.,.5],
                [1/3,1/3,.5],
                [.5,0.,.5],
                [0.,0.,.5]])
X = ['G','K','M','G','A','H','L','A']

atoms = read('scf.pwo')
f = open( seedname + '.win', 'w')

def proj_set(mode='projection'):
    print('num_bands\t=\t%d' % num_bands,file=f)
    print('num_wann\t=\t%d' % num_wann,file=f)
    print('num_iter\t=\t200' ,file=f)
    print('dis_num_iter\t=\t%d' % dis_num_iter,file=f)
    print('kmesh_tol\t=\t0.0001' ,file=f)
    print(file=f)
    if mode == 'scdm':
        print('auto_projections = .true.',file=f)
    elif mode == 'projection':
        print('dis_win_max\t=\t%.1f' % dis_win_max, file=f)
        print('begin projections',file=f)
        print('C:pz',file=f)
        print('end projections',file=f)
    print(file=f)

def band_plot():
    print('write_xyz = .true.',file=f)
    print('bands_plot = .true.',file=f)
    print('bands_num_points = 100',file=f)
    print('bands_plot_format = gnuplot',file=f)
```

```
    print(file=f)
    print('begin kpoint_path',file=f)
    for i in range(len(X)-1):
        print('%s%10.5f%10.5f%10.5f   %s%10.5f%10.5f%10.5f' % \
(X[i],kpath[i][0],kpath[i][1],kpath[i][2],X[i+1],kpath[i+1][0],kpath[i+1][1],kpath[i+1][2]),file=f)
    print('end kpoint_path',file=f)
    print(file=f)

def atoms_pos():
    natom = atoms.get_global_number_of_atoms()
    atom_symbols = atoms.get_chemical_symbols()
    print('begin atoms_cart',file=f)
    for i in range(natom):
        print(atoms.get_chemical_symbols()[i],end='\t',file=f)
        for j in range(3):
            print('%12.8f' % atoms.positions[i][j],end='\t',file=f)
        print(file=f)
    print('end atoms_cart',file=f)
    print(file=f)

def unit_cell():
    print('begin unit_cell_cart',file=f)
    for i in range(3):
        for j in range(3):
            print('%12.8f' % atoms.cell[i][j],end='\t',file=f)
        print(file=f)
    print('end unit_cell_cart',file=f)
    print(file=f)

def kpoints():
    print('mp_grid:  %d  %d  %d' % (kpts[0],kpts[1],kpts[2]),file=f)
    print(file=f)
    print('begin kpoints',file=f)
    for i in range(kpts[0]):
        for j in range(kpts[1]):
            for k in range(kpts[2]):
                print('%12.8f%12.8f%12.8f' % (i/kpts[0],j/kpts[1],k/kpts[2]),file=f)
    print('end kpoints',file=f)

def tb_para():
    print('write_hr = .true.',file=f)
    print('write_tb = .true.',file=f)
    print('write_hr_diag = .true.',file=f)
    print(file=f)

if __name__=="__main__":
    proj_set()
    band_plot()
    tb_para()
    atoms_pos()
    unit_cell()
    kpoints()
    f.close()
```

while a templete `pw2wan.inp` file is:

```
&inputpp
   outdir = '../tmp'
   prefix = 'graphene'
```

```
   seedname = 'graphene'
   spin_component = 'none'
   write_mmn = .true.
   write_amn = .true.
/
```

4.2 Graphene

For graphene case, we want to emphasize that `assume_isolated = '2D'` is very helpful.
After obtaining the wannier functions, we get the DFT band structure data by a python script `QE_band.py`:

```
from ase.calculators.espresso import Espresso
from ase.io import read
import os
import numpy as np

seed = 'graphene'
kpath = 'GKMG'
nbands = 16
npoints = int([line for line in open(seed+'_band.kpt')][0])
ef = float([line for line in open('scf.pwo') if 'Fermi' in line][0].split()[-2])
with open('FERMI_ENERGY','w') as f:
    print('# Fermi Energy in scf calculation',file=f)
    print('%10.6f' % ef,file=f)

atoms = read('../POSCAR')
pseudo_dir='/opt/LDA'
pseudopotentials={'C':'C.pz-n-kjpaw_psl.1.0.0.UPF'}

kpts={'path':kpath, 'npoints':npoints}
path = atoms.cell.bandpath(kpath,npoints=npoints)
x, X, _ = path.get_linear_kpoint_axis()
with open('kpath.dat','w') as f:
    for k in x:
        print(k,file=f)
with open('highk.dat','w') as f:
    for k in X:
        print(k,file=f)

def calc_qe(mode='bands',ecut=60,conv=1e-12):
    calc = Espresso(calculation=mode,
                label=mode,
                outdir='./tmp',
                prefix='graphene',
                verbosity='high',
                pseudo_dir=pseudo_dir,
                pseudopotentials=pseudopotentials,
                kpts=kpts,
                nbnd=nbands,
                occupations='smearing',
                smearing='gauss',
                degauss=1e-1,
                ecutwfc=ecut,
                ecutrho=8*ecut,
                ibrav=4,
                assume_isolated = '2D',
                conv_thr=conv)
    return calc
```

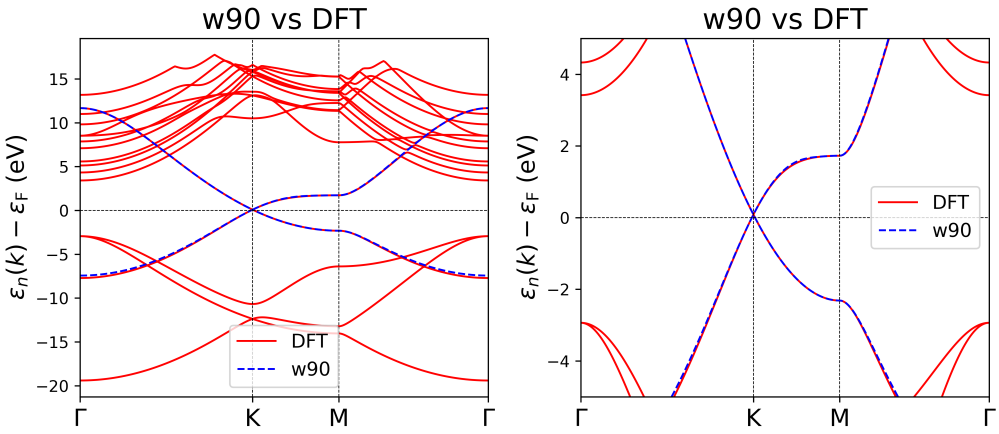

Figure 5: QE band structure vs wannier p_z interpolated band structure of graphene

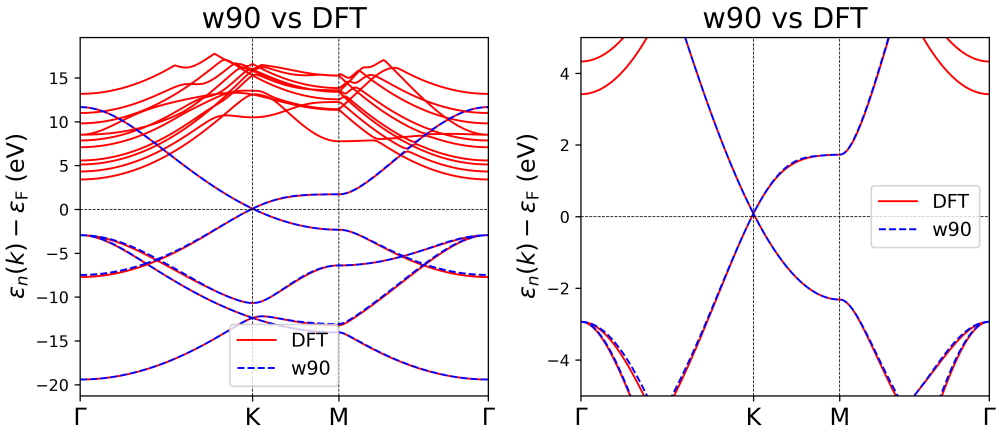

Figure 6: QE band structure vs wannier $p_z + \sigma$ interpolated band structure of graphene

```
calc = calc_qe()
atoms.calc = calc
calc.calculate(atoms)
e_nk = calc.band_structure().energies[0].T - ef
np.savetxt('e_nk.dat', e_nk)
```

and plot the band structure by a python script, which is almost the same as `w90_vs_VASP.py` in VASP case. The plotted band structures of p_z and $p_z + \sigma$ configuration are shown in Figure. 5 and 6, respectively.

4.3 Graphite

In graphite case, the main difference with graphene case is the `kpoints` and no `assume_isolated` in scf calculation. The plotted band structures of p_z and $p_z + \sigma$ configuration are shown in Figure. 7 and 8, respectively.

5 GPAW + wannier90

5.1 General steps

Generally, we need the following 4 steps:

- do high precision (`convergence={'density':1e-6}` is usually higher than `energy` 1e-8) scf calculation, while tune off symmetry and set number of bands.
- generate `win`, `nnkp`, `amn`, `mmn`, `eig` files via `gpaw.wannier90` module. (Only serial mode is supported)

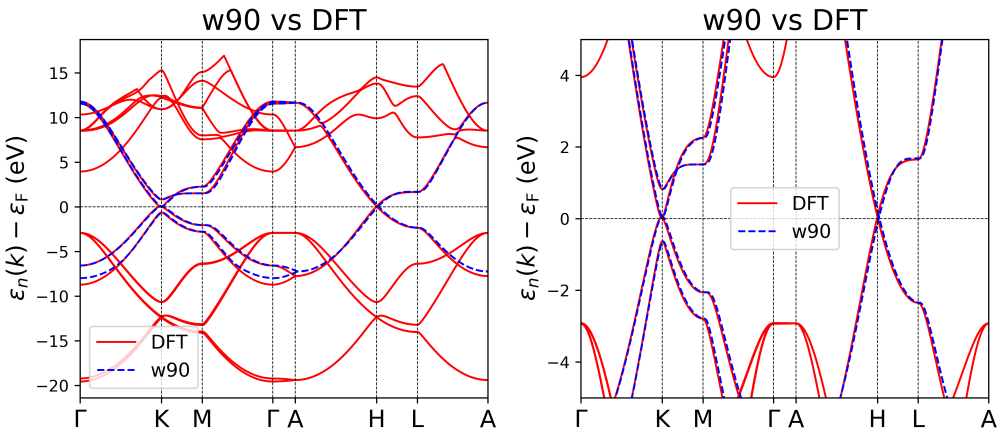

Figure 7: QE band structure vs wannier p_z interpolated band structure of graphite

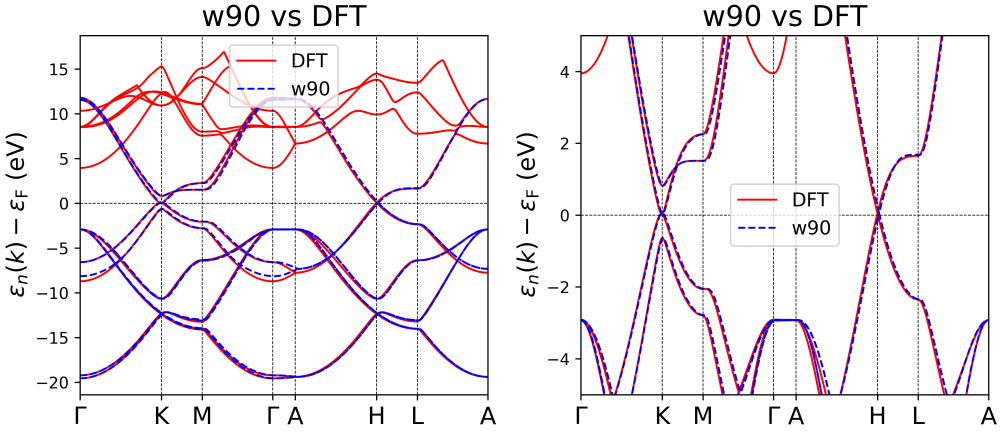

Figure 8: QE band structure vs wannier $p_z + \sigma$ interpolated band structure of graphite

- edit the `win` file according to our demand.
- do `wannier90.x` calculation and plot the results.

5.2 Graphite

In graphite case, we do the high precision scf calculation by a script `GPAW_scf.py`:

```
from gpaw import GPAW, PW, FermiDirac
from ase.io import read

atoms = read('../POSCAR')
calc = GPAW(mode=PW(520),
            xc='LDA',
            kpts=(15,15,3),
            occupations=FermiDirac(0.01),
            convergence={'density':1e-6},
            txt='scf.out')
atoms.calc = calc
atoms.get_potential_energy()
calc.write('scf.gpw')
calc.fixed_density(
            symmetry='off',
            nbands=20,
            convergence={'bands':16},
            txt='nscf.out').write('nscf.gpw','all')
```

we utilize the `gpaw.wannier90` module by script `GPAW_wann.py`:

```
import os
import gpaw.wannier90 as w90
from gpaw import GPAW, mpi

seed = 'gra'
orbitals_ai = [[2],[2],[2],[2]]

if __name__=="__main__":
    calc = GPAW('nscf.gpw')
    w90.write_input(calc,orbitals_ai=orbitals_ai,
            bands=range(2,16),
            seed = seed,
            num_iter = 200,
            dis_num_iter = 1e4,
            write_xyz=True)
    os.system('wannier90.x -pp ' + seed)

    w90.write_projections(calc,orbitals_ai=orbitals_ai,seed=seed)
    w90.write_eigenvalues(calc,seed=seed)
    w90.write_overlaps(calc,seed=seed)
```

we get the band structure data of DFT by script `GPAW_band.py`:

```
from gpaw import GPAW, PW, FermiDirac, mpi
import numpy as np
from ase.io import read
from ase.parallel import paropen
from GPAW_wann import seed

## band structure calculation
npoints = int([line for line in open(seed+'_band.kpt')][0])
atoms = read('scf.out')
```

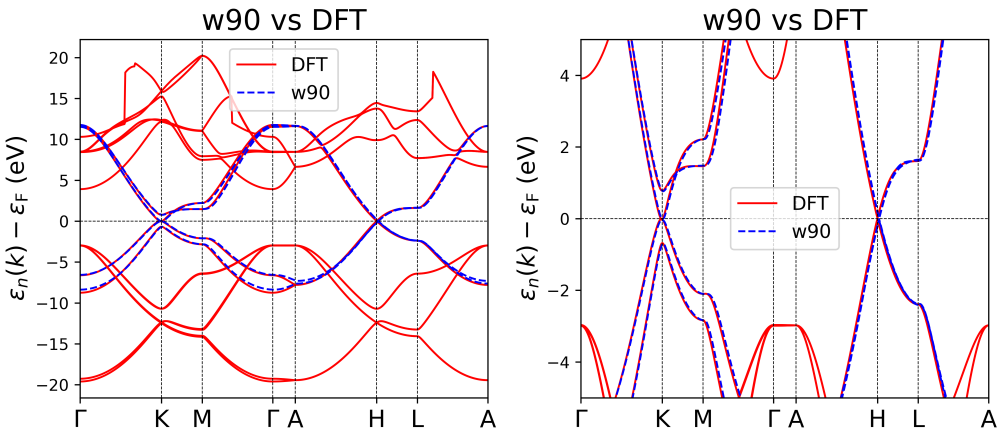

Figure 9: GPAW band structure vs wannier p_z interpolated band structure of graphite

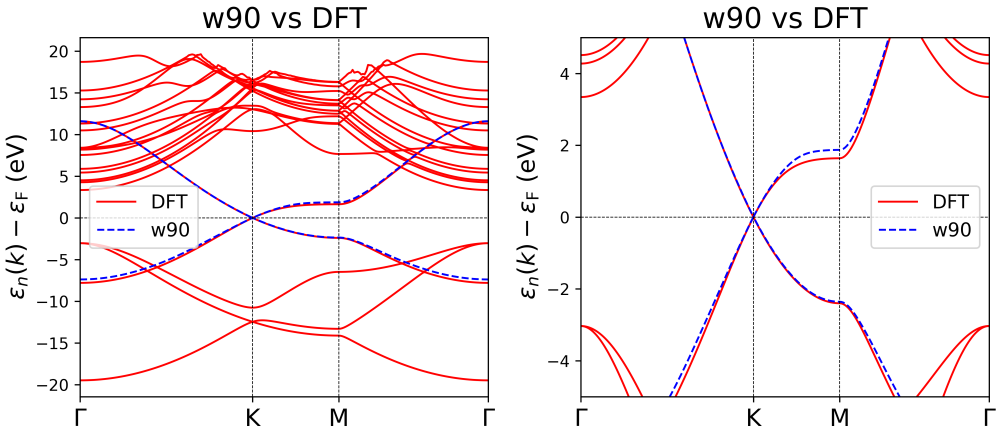

Figure 10: GPAW band structure vs wannier p_z interpolated band structure of graphene

```
path = atoms.cell.bandpath('GKMGAHLA',npoints=npoints)
calc = GPAW('scf.gpw').fixed_density(symmetry='off',
                        kpts=path.kpts,
                        nbands=16,
                        txt='band.out')
## save the data of band structure
x, X, _ = path.get_linear_kpoint_axis()
with paropen('kpath.dat','w') as f:
    for k in x:
        print(k,file=f)

with paropen('highk.dat','w') as f:
    for k in X:
        print(k,file=f)

ef = calc.get_fermi_level()
e_nk = calc.band_structure().energies[0].T - ef
if mpi.world.rank == 0:
    with open('FERMI_ENERGY','w') as f:
        print('# Fermi level in scf calculation',file=f)
        print('%10.6f' % ef,file=f)
    np.savetxt('e_nk.dat', e_nk)
```

and plot the band struture by script `w90_vs_GPAW.py`:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.use('Agg')
from GPAW_wann import seed, orbitals_ai
from gpaw import GPAW

nwan = 0
for orbital in orbitals_ai:
    nwan += len(orbital)

lw = 1.2
fontsize = 15
title = 'w90 vs DFT'

ef = np.loadtxt('FERMI_ENERGY')
kx = np.loadtxt('kpath.dat')
X = np.loadtxt('highk.dat')
e_nk = np.loadtxt('e_nk.dat')
klabels = [r'$\Gamma$','K','M',r'$\Gamma$','A','H','L','A']

## read band data from _band.dat file
ewan_kn=np.zeros((len(kx),nwan),dtype=float)
f=open(seed+'_band.dat')
for i in range(nwan):
    for j in range(len(kx)):
        l=f.readline()
        ewan_kn[j,i]=float(l.split()[1]) - ef
    l=f.readline()
f.close()
ewan_nk = ewan_kn.T

## plot the bandstructrue
def plot_band(window=False):
```

```
    for e_k in e_nk:
        plt.plot(kx,e_k,c='r',lw=1.2)
    plt.plot(-1,-1,c='r',lw=1.2,label='DFT')
    for ewan_k in ewan_nk:
        plt.plot(kx,ewan_k,c='b',lw=1.2,ls='--')
    plt.plot(-1,-1,c='b',lw=1.2,ls='--',label='w90')
    for k in X[1:-1]:
        plt.axvline(x=k,c='k',lw=0.5,ls='--')
    plt.axhline(y=0,c='k',lw=0.5,ls='--')
    plt.xticks(X,klabels,size=15)
    plt.yticks(size=10)
    plt.ylabel(r'$\varepsilon_n(k) - \varepsilon_{\mathrm{F}}$ (eV)',size=15)
    plt.title(title, size=18)
    plt.xlim([0,kx[-1]])
    plt.legend(fontsize=12)
    if window:
        plt.axis([0,kx[-1],-5,5])

plt.figure(figsize=(9,4))
plt.subplot(121)
plot_band()
plt.subplot(122)
plot_band(True)
plt.tight_layout()
plt.savefig('GPAW_graphite_pz.png',dpi=600)
plt.close()
```

The plotted band structure is shown in Figure. 9. The data of wannier function centre and spread is shown in Table 2.

Table 2: Comparison of wannier function centre and spread of graphite p_z case from different DFT code

orbital	WF centre (Cartesian coordinates in Å)			spread (Å^2)
atom C1	0.00000	0.00000	0.00000	
atom C2	0.00000	1.41309	0.00000	
atom C3	0.00000	1.41309	3.32540	
atom C4	1.22377	0.70654	3.32540	
VASP C1	0.000002	-0.000006	0.000021	0.95723897
VASP C2	0.000001	1.413085	-0.000027	0.95332544
VASP C3	0.000006	1.413066	3.325431	0.95332542
VASP C4	1.223778	0.706522	3.325383	0.95713514
QE C1	0.000000	0.000000	0.000000	0.96971893
QE C2	0.000000	1.413089	0.000000	0.96304329
QE C3	0.000000	1.413089	3.325396	0.96304307
QE C4	1.223770	0.706544	3.325396	0.96971961
GPAW C1	-0.000151	-0.001175	-0.000080	0.97871263
GPAW C2	0.021509	1.332993	-0.000362	1.70440385
GPAW C3	0.022487	1.332452	3.229416	2.52595809
GPAW C4	1.142714	0.686301	3.209020	2.52686018

5.3 Graphene

In graphene case, the main difference is that `kpts` becomes (15,15,1), and the plotted band structure is shown in Figure. 10. We notice that, the quality of wannier functions generated from GPAW is not as good as those of VASP and QE, from the data of wannier function centre and spread, as shown in Table 3.

Table 3: Comparison of wannier function centre and spread of graphene p_z case from different DFT code

orbital	WF centre (Cartesian coordinates in Å)			spread (Å^2)
atom C1	0.00000	0.00000	10.00000	
atom C2	1.22363	0.70646	10.00000	
VASP C1	0.000006	0.000018	10.000021	0.96983756
VASP C2	1.223643	0.706481	10.000083	0.96983504
QE C1	0.000000	-0.000000	9.999999	0.95023809
QE C2	1.223630	0.706463	9.999999	0.95023873
GPAW C1	-0.000683	-0.000762	10.000000	0.88684912
GPAW C2	1.144137	0.679121	10.000000	2.04059119

References

[1] Nicola Marzari, Arash A Mostofi, Jonathan R Yates, Ivo Souza, and David Vanderbilt. Maximally localized wannier functions: Theory and applications. *Reviews of Modern Physics*, 84(4):1419, 2012.

[2] Arash A Mostofi, Jonathan R Yates, Young-Su Lee, Ivo Souza, David Vanderbilt, and Nicola Marzari. wannier90: A tool for obtaining maximally-localised wannier functions. *Computer physics communications*, 178(9):685–699, 2008.

[3] Georg Kresse and Jürgen Hafner. Ab initio molecular dynamics for liquid metals. *Physical review B*, 47(1):558, 1993.

[4] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, et al. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of physics: Condensed matter*, 21(39):395502, 2009.

[5] Jussi Enkovaara, Carsten Rostgaard, J Jørgen Mortensen, Jingzhe Chen, M Dułak, Lara Ferrighi, Jeppe Gavnholt, Christian Glinsvad, V Haikola, HA Hansen, et al. Electronic structure calculations with gpaw: a real-space implementation of the projector augmented-wave method. *Journal of physics: Condensed matter*, 22(25):253202, 2010.