TB2J

Xu He

Jun 20, 2024

CONTENTS:

TB2J is an open-source Python package for the automatic computation of magnetic interactions (including exchange and Dzyaloshinskii-Moriya) between atoms of magnetic crystals from density functional Hamiltonians based on Wannierfunctions or linear combinations of atomic orbitals. The program is based on Green's function method with the local rigid spin rotation treated as a perturbation. As input, the package uses the output of either Wannier90, whichis interfaced with many density functional theory packages, or of codes based on localized orbitals (SIESTA, OpenMX and Abacus). A minimal user-input is needed, which allows for easy integration into high-throughput workflows.

The TB2J project is initialized in the PhyTheMa and Nanomat teams in the University of Liege.

The source code can be found at https://github.com/mailhexu/TB2J.

For questions please use the online forum at https://groups.google.com/g/tb2j, or send email to mailto:tb2j@googlegroup.com.

More TB2J examples with full DFT/Wannier data can be found at https://github.com/mailhexu/TB2J_examples .

There are video tutorials in TB2J channel on youtube: https://www.youtube.com/channel/UCPbmKE10Wz3orbo4x-g0c9A .

CHAPTER

ONE

INSTALLATION

1.1 Dependencies

TB2J is a python package which requires python version higher than 3.6 to work. It depends on the following packages.

- numpy>1.16.5
- scipy
- matplotlib
- ase>=3.19
- tqdm>=4.42.0
- p_tqdm
- pathos
- packaging

If you use pip to intall, they will be automatically installed so there is no need to install them manually before installing TB2J.

There are some optional dependencies, which you need to install if needed.

- sisl>0.10.0 (optional) for Siesta interface
- GPAW (optional) For gpaw interface (not yet fully operational).

1.2 How to install

The most easy way to install TB2J is to use pip:

```
pip install TB2J
```

You can also download TB2J from the github page, and install with

```
python setup.py install
```

The –user option will help if there is permission problem.

It is suggested that it being installed within a virtual environment using e.g. pyenv or conda.

By default, TB2J only forces the non-optional dependencies to be installed automatically. The sisl package which is used to read the Hamiltonian from the Siesta or OpenMX output is needed, which can also be installed with pip. The

GPAW-TB2J interface is through python directly, which of course requires the gpaw python package. The sisl and gpaw python package can be installed via pip, too. For example:

```
pip3 install sisl
```

1.2.1 How to install in a virtual environment

It is recommend to install TB2J in virtual enviroment (venv), which is like a world parallel to the main python environment where the other packages are installed. With this, conflictions between library versions can be avoided. For example, may be you have other packages only work with a old version of numpy, whereas TB2J requires a newer version. Then within this venv you can have the new version without needing to worry about the conflictions.

One way to build a python venv is to use venv library built in python. We can make a new python virtual environment named TB2J like this:

```
python3 -m venv <your path>/TB2J
```

where you can replace <your path> to the path where you want to put the files for the venv.

Then you can activate the venv py using

```
source <your path>/TB2J/bin/activate
```

The within this venv you can install the python packages. And this venv should be activated when you use TB2J.

There are other ways to build virtual environments, for example, with conda .

CHAPTER

TWO

CONVENTIONS OF HEISENBERG MODEL

Before you use the TB2J output, please read very carefully this section. There are many conventions of the Heisenberg. We strongly suggest that you clearly specify the convention you use in any published work. Here we describe the convention used in TB2J.

The Heisenberg Hamiltonian contains four different parts and reads as

$$\begin{aligned} E = & -\sum_i K_i \vec{S}_i^2 \\ & -\sum_{i \neq j} \Big[J_{ij}^{iso} \vec{S}_i \cdot \vec{S}_j \\ & + \vec{S}_i \mathbf{J}_{ij}^{ani} \vec{S}_j \\ & + \vec{D}_{ij} \cdot \left(\vec{S}_i \times \vec{S}_j \right) \Big], \end{aligned}$$

where the first term represents the single-ion anisotropy (SIA), the second the isotropic exchange, and the third term is the symmetric anisotropic exchange, where $\mathbf{J}^{ani}$ is a 3×3 tensor with $J^{ani} = J^{ani,T}$. The final term is the DMI, which is antisymmetric. Importantly, the SIA is not accessible from Wannier 90 as it requires separately the spin-orbit coupling part of the Hamiltonian. However, it is readily accessible from constrained DFT calculations.

We note that there are several conventions for the Heisenberg Hamiltonian, here we take a commonly used one in atomic spin dynamics: we use a minus sign in the exchange terms, i.e. positive exchange J values favor ferromagnetic alignment. Every pair ij is taken into account twice, J_{ij} and J_{ji} are both in the Hamiltonian. Similarly, both uv and vu are in the symmetric anisotropic term. The spin vectors $\vec{S}_i$ are normalized to 1, so that the parameters are in units of energy. The other commonly used conventions differ in a prefactor 1/2 or a summation over different ij pairs only. The conversion factors to other conventions are given in the following table. For other conventions in which the spins are not normalized, the parameters need to be divided by $|\vec{S}_i \cdot \vec{S}_j|$ in addition.

CHAPTER

THREE

TUTORIAL

3.1 Use TB2J with Wannier90

This tutorial uses cubic $SrMnO_3$ as an example to show how to calculate the exchange parameters for the Heisenberg model starting from density functional theory. First, the Hamiltonian in the basis of Wannier functions (WF) is constructed using Wannier90. Then, TB2J is used to calculate the exchange parameters. We assume that the reader has a basic knowledge of maximally localized WFs and the Wannier90 package (see Maximally localized Wannier Functions, Wannier90).

Before beginning, you might consider to work in a subdirectory for this tutorial. Why not Work_tb2j?

The input files for the tutorial can be found inside examples/abinit-w90/SrMnO3 in your TB2J directory. Please copy abinit.in, abinit.files and the three pseudopotential files (inside the psp directory) to *Work_tb2j*. You also need the two files abinito_w90_down.win and abinito_w90_up.win which provide additional input for Wannier90. The names of these two files are _w90_.win with the prefix being given in the forth line of the .files file. Modify the .files file such that the entries match the location of your files.

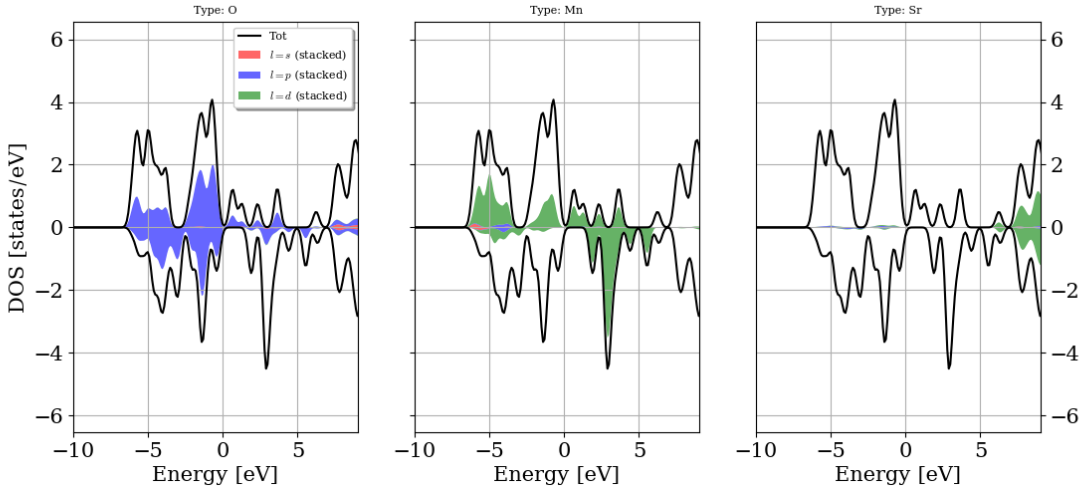
3.1.1 Step 0: Find the orbitals and energy range to be used in the Wannier Function Hamiltonian.

Before we can construct the Hamiltonian in the basis of the Wannier functions, we need to determine which orbitals to include in the construction. We need to include the orbitals with energies around the Fermi energy (E_F). Since we are interested in calculating exchange parameters we need to include spin as a degree of freedom in the calculation and select the magnetic orbitals and all orbitals that overlap with them. To determine the orbitals and energy range, we calculate either the density of states or the band structure of the system. For $SrMnO_3$ the density of states is given in the figure below.

As we can see, the Mn 3d and O 2p orbitals should be included into the WF Hamiltonian. The Sr 4d orbitals are too high in energy, so we exclude them from the WF Hamiltonian.

3.1.2 Step 1: Construct WF Hamiltonian from DFT.

The Wannier90 code makes use of two energy windows to disentangle the bands. An outer window (the disentangle window), which contains all the required orbitals, and an inner window (the frozen window), which only contain the required orbitals, should be provided. From the DOS we find that all the Mn 3d and O 2p bands are between -10 and 10 eV, the Sr 4d bands above 6 eV, which should be excluded from the frozen window. Thus we can select the energy window (-10, 10) eV and the frozen window of (-9, 5) eV. Note that the energy defined in Wannier90 is not relative to E_F, so we need to add the Fermi energy (here: 6.15 eV) to the energies. We use Mn d and O p orbitals as an initial guess for the WFs. This information can be found in the .win files




```
# Energy windows (Fermi energy is 6.15 eV)
dis_win_min = -3.85
dis_win_max = 16.15
dis_froz_min = 1.15
dis_froz_max = 11.15

begin projections
Mn: d
O : p
end projections
```

For a detailed explanation of the input variables for Wannier90 please see Wannier90. For our purpose, it is important to write out the Hamiltonian and the centers of the Wannier functions.

```
# write the postitions of WF
write_xyz = true

# write the WF Hamiltonian (Note for W90 version<2.1, it is hr_plot)
write_hr = true
```

Alternatively, the Wannier hamiltonian and the position operator can be written into one "_tb.dat" file, which can be read by TB2J since version 0.8.2

```
# write the WF Hamiltonian and the position operator
write_tb=ture
```

The following lines need to be added to the abinit input file to generate WFs.

```
prtwant 2    # enable wannier90
w90iniprj 2  # use projection to orbitals instead of random.
w90prtunk 0  # use 1 if you want to visualize the WF's later.
```

Now you can run

```
abinit < abinit.files > log 2> err
```

which generates the files below for spin up, and the same set for spin down

```
abinito_w90_up_hr.dat abinito_w90_up_centres.xyz   abinito_w90_up.wout
```

The .dat file contains the Hamiltonian, the .xyz file contains the Wannier centers. The .wout file has a summary of the process of running Wannier90 and will be used to calculate the exchange parameters.

If you're using Wannier90 version < 3.0, the spin down files are not automatically generated due to a bug. To get the files, the following command is needed:

```
wannier90.x abinito_w90_down
```

To get localized WFs can be tricky sometimes. It is necessary to check if the WFs are localized by looking at the .wout file. For example, we have

```
Final State
 WF centre and spread    1  (  1.904992,  1.904992,  1.904992 )     0.50185811
 WF centre and spread    2  (  1.904992,  1.904992,  1.904992 )     0.48650086
 WF centre and spread    3  (  1.904992,  1.904992,  1.904992 )     0.48650086
 WF centre and spread    4  (  1.904992,  1.904992,  1.904992 )     0.50185997
 WF centre and spread    5  (  1.904992,  1.904992,  1.904992 )     0.48650084
 WF centre and spread    6  (  1.904992,  1.904992, -0.000000 )     0.74591265
 WF centre and spread    7  (  1.904992,  1.904992,  0.000000 )     0.96557405
 WF centre and spread    8  (  1.904992,  1.904992, -0.000000 )     0.96557405
 WF centre and spread    9  ( -0.000000,  1.904992,  1.904992 )     0.96557489
 WF centre and spread   10  ( -0.000000,  1.904992,  1.904992 )     0.74589254
 WF centre and spread   11  (  0.000000,  1.904992,  1.904992 )     0.96557379
 WF centre and spread   12  (  1.904992,  0.000000,  1.904992 )     0.96557489
 WF centre and spread   13  (  1.904992, -0.000000,  1.904992 )     0.96557379
 WF centre and spread   14  (  1.904992, -0.000000,  1.904992 )     0.74589254
 Sum of centres and spreads ( 20.954915, 20.954915, 20.954915 )    10.49436382
```

Usually, 3d orbitals have a spread of less than 1 , and the O 2p orbitals have a spread of less than 2 .

3.1.3 Step 2: Run TB2J

Before running TB2J, an extra file, which contains the atomic structure, needs to be prepared. It can be either a VASP POSCAR file. (For abinit, the abinit.in file is also fine if no fancy feature is used, like use of *, or units. POSCAR files are recommended because they are simple. Note that the file extension are used to identify the format, for example, Quantum ESPRESSO input should be name with *.pwi) The supported file format are can be found on the list in: https://wiki.fysik.dtu.dk/ase/ase/io/io.html

(From version 0.6.2 this file is no more necessary as TB2J can read the atomic structures from the Wannier90 .win file). The –posfile option will still be used by default if it is specified.)

With the WF Hamiltonian generated, we can calculate the exchange parameters now. In the scripts directory inside your TB2J directory you find the wann2J.py script. Please make sure that it is executable and issue the command

```
wann2J.py --posfile abinit.in --efermi 6.15 --kmesh 4 4 4 --elements Mn --prefix_up␣
↪abinito_w90_up --prefix_down abinito_w90_down  --emin -10.0 --emax 0.0
```

The parameters are:

- efermi: Fermi energy in eV
- kmesh: k-point mesh. Default is 5 5 5

- elements: the magnetic elements
- prefix_up: prefix for spin up channel of the Wannier90 output
- prefix_down: prefix for spin down channel of Wannier90 output.
- emin: the lower limit of the electron energy. (in eV, relative to Fermi energy.)
- emax: the upper limit of the electron energy. Should be zero. (Note: this parameter is no more useful will be deprecated soon).

Now we should have the files containing the J parameters in the TB2J_results directory.

```
TB2J_results/
├── exchange.txt
├── Multibinit
│   ├── exchange.xml
│   ├── mb.files
│   └── mb.in
├── TomASD
│   ├── exchange.exch
│   └── exchange.ucf
└── Vampire
    ├── input
    ├── vampire.mat
    └── vampire.UCF
```

- exchange.txt: A human readable file.
- Multibinit directory: the files file, input file and xml file, which can be used as templates to run spin dynamics in Multibinit.
- The input for a few spin dynamics codes (Tom's ASD, and Vampire) are also included.

3.1.4 Noncollinear calculation

For calculations with non-collinear spin, the –spinor option should be used. It is also necessary to specify whether in the Hamiltonian the order of the basis, either group by spin (orb1_up, orb2_up, … orb1_down, orb2_down, …) or by orbital (orb1_up, orb1_down, orb2_up, orb2_down,…), with the –groupby option (either spin or orbital). The –prefix_spinor option is used to specify the prefix of the Wannier90 outputs. Here is an example of the command:

```
wann2J.py --spinor --groupby spin --posfile abinit.in --efermi 6.15 --kmesh 4 4 4 --
↪elements Mn --prefix_spinor abinito
```

3.2 Use TB2J with Siesta

In this tutorial we will learn how to use TB2J with Siesta. First we calculate the isotropic exchange in bcc Fe. Then we calculate the isotropic exhange, anisotropic exchange and Dzyanoshinskii-Moriya interaction (DMI) parameters from a calculation with spin-orbit coupling enabled. Before running this tutorial, please make sure that the sisl package is installed.

3.2.1 Collinear calculation without SOC

Let's start from the example of BCC Fe. The input files used can be found in the examples/Siesta/bccFe directory.

First, we do a siesta self consistent calculation with the BCC Fe primitive cell of bcc Fe has only one Fe atom. In the example, we use the pseudopotential from the PseudoDojo dataset in the psml format. Note that at the moment the psml support is implemented in the master development and "Max" branches of siesta (see https://gitlab.com/siesta-project/siesta/-/wikis/Guide-to-Siesta-versions for the different versions of siesta, and https://gitlab.com/siesta-project/siesta/-/wikis/How-to-build-the-master-version-of-Siesta for how to build it.). We need to save the electronic Kohn-Sham Hamiltonian in the atomic orbital basis set with the options:

```
SaveHS True
```

and this option to use the netcdf format (if netcdf is enabled within the siesta version being used).

::

    CDF.Save True

After that, we will have the files siesta.nc and DMHS.nc file, which contains the Hamiltonian and overlap matrix information.

Now we can run the siesta2J.py command to calculate the exchange parameters:

```
siesta2J.py --fdf_fname siesta.fdf --elements Fe --kmesh 7 7 7
```

This first read the siesta.fdf, the input file for Siesta. It then read the Hamiltonian and the overlap matrices, calculate the J with a $7 \times 7 \times 7$ k-point grid. This allows for the calculation of exchange between spin pairs between i and j in a $7 \times 7 \times 7$ supercell, where i is fixed in the center cell. Note: the kmesh is not dense enough for a practical calculation. Please check the convergence.

3.2.2 Non-collinear calculation

The anisotropic exchange and the DMI parameters can be calculated with non-collinear DFT calculation. The procedure is almost the same as in the collinear calculation except that the parameters for non-collinear calculation must be set in the Siesta input (Spin should be set to non-colinear or spin-orbit).

3.3 Use TB2J with OpenMX

In this tutorial we will learn how to use TB2J with OpenMX with the example of cubic SrMnO3. The example input files can be found in the examples directory of

> The interface to OpenMX is distributed as a plugin to TB2J called TB2J OpenMX under the GPL license, which need to be installed separately, because code from OpenMX which is under the GPL license is used in the parser of OpenMX files.

3.3.1 Install TB2J-OpenMX

```
pip install TB2J-OpenMX
```

3.3.2 running TB2J

In the DFT calculation, the "HS.fileout on" options should be enabled, so that the Hamiltonian and the overlap matrices are written to a ".scfout" file. Then we can run the command openmx2J.py. The necessary input are the path of the calculation, the prefix of the OpenMX files, and the magnetic elements:

```
openmx2J.py -- prefix openmx --elements Fe --kmesh 7 7 7
```

openmx2J.py then read the openmx.xyz and the openmx.scfout files from the OpenMX output, and output the results to TB2J_results. Note: the kmesh is not dense enough for a practical calculation.

3.4 Use TB2J with ABACUS

In this tutorial we will learn how to use TB2J with ABACUS. The TB2J-ABACUS interface is available since TB2J version 0.8.0. There are three types of basis set in ABACUS, the plane-wave (PW), the linear-combinatio of atomic orbitals (LCAO), and the LCAO-in-PW. With the LCAO basis set, TB2J can directly take the output and compute the exchange parameters. For the other type of basis set, the Wannier90 interace can be used instead. In this tutorial we will use LCAO.

3.4.1 Collinear calculation without SOC

Let's start from the example of Fe. The example files can be found here: https://github.com/mailhexu/TB2J_examples/tree/master/Abacus/Fe_no_SOC .

First do the ABACUS calculation. Note that the Kohn-Sham Hamiltonian and the overlap matrix is needed as the input to TB2J. We need to put

```
out_mat_hs2  1
```

in the ABACUS INPUT file, so that the Hamiltonian matrix H(R) (in Ry) and overlap matrix S(R) will be written into files in the directory `OUT.${suffix}` . In the INPUT, the line

```
sufffix Fe
```

specifies the suffix of the output. Thus the output will be in the directory OUT.Fe when the DFT calculation is finished.

In this calculation, we set the path to the directory of the DFT calculation, which is the current directory (". ") and the suffix to Fe.

Now we can run the abacus2J.py command to calculate the exchange parameters:

```
abacus2J.py --path . --suffix Fe --elements Fe  --kmesh 7 7 7
```

This first read the atomic structures from th STRU file, then read the Hamiltonian and the overlap matrices stored in the files named starting from "data-HR-" and "data-SR-" files. It also read the fermi energy from the OUT.Fe/running_scf.log file.

With the command above, we can calculate the J with a 7x7x7 k-point grid. This allows for the calculation of exchange between spin pairs between 7x7x7 supercell. Note: the kmesh is not dense enough for a practical calculation. For a

very dense k-mesh, the –rcut option can be used to set the maximum distance of the magnetic interactions and thus reduce the computation cost. But be sure that the cutoff is not too small.

3.4.2 Non-collinear calculation with SOC

The DMI and anisotropic exchange are result of the SOC, therefore requires the DFT calculation to be done with SOC enabled. To get the full set of exchange parameters, a "rotate and merge" procedure is needed, in which several DFT calculations with either the structure or the spin rotated are needed. For each of the non-collinear calcualtion, we compute the exchange parameters from the DFT calculation with the same command as in the collienar case.

```
abacus2J.py --path . --suffix Fe --elements Fe  --kmesh 7 7 7
```

And then the "TB2J_merge.py" command can be used to get the final spin interaction parameters.

3.4.3 Parameters of abacus2J.py

We can use the command

```
abacus2J.py --help
```

to view the parameters and the usage of them in abacus2J.py.

```
TB2J version 0.8.0
Copyright (C) 2018-2024  TB2J group.
This software is distributed with the 2-Clause BSD License, without any warranty. For␣
↪more details, see the LICENSE file delivered with this software.


usage: abacus2J.py [-h] [--path PATH] [--suffix SUFFIX] [--elements [ELEMENTS ...]] [--
↪rcut RCUT] [--efermi EFERMI]
                   [--kmesh [KMESH ...]] [--emin EMIN] [--use_cache] [--nz NZ] [--cutoff␣
↪CUTOFF]
                   [--exclude_orbs EXCLUDE_ORBS [EXCLUDE_ORBS ...]] [--np NP] [--
↪description DESCRIPTION]
                   [--orb_decomposition] [--fname FNAME] [--output_path OUTPUT_PATH]

abacus2J: Using magnetic force theorem to calculate exchange parameter J from ABACUS␣
↪Hamiltonian in the LCAO mode

options:
  -h, --help            show this help message and exit
  --path PATH           the path of the ABACUS calculation
  --suffix SUFFIX       the label of the ABACUS calculation. There should be an output␣
↪directory called OUT.suffix
  --elements [ELEMENTS ...]
                        list of elements to be considered in Heisenberg model.
  --rcut RCUT           range of R. The default is all the commesurate R to the kmesh
  --efermi EFERMI       Fermi energy in eV. For test only.
  --kmesh [KMESH ...]   kmesh in the format of kx ky kz. Monkhorst pack. If all the␣
↪numbers are odd, it is Gamma
                        cenetered. (strongly recommended), Default: 5 5 5
```

(continues on next page)

(continued from previous page)

```
  --emin EMIN           energy minimum below efermi, default -14 eV
  --use_cache           whether to use disk file for temporary storing wavefunctions and␣
↪hamiltonian to reduce memory
                        usage. Default: False
  --nz NZ               number of integration steps. Default: 50
  --cutoff CUTOFF       The minimum of J amplitude to write, (in eV). Default: 1e-7 eV
  --exclude_orbs EXCLUDE_ORBS [EXCLUDE_ORBS ...]
                        the indices of wannier functions to be excluded from magnetic␣
↪site. counting start from 0.
                        Default is none.
  --np NP               number of cpu cores to use in parallel, default: 1
  --description DESCRIPTION
                        add description of the calculatiion to the xml file. Essential␣
↪information, like the xc
                        functional, U values, magnetic state should be given.
  --orb_decomposition   whether to do orbital decomposition in the non-collinear mode.␣
↪Default: False.
  --fname FNAME         exchange xml file name. default: exchange.xml
  --output_path OUTPUT_PATH
                        The path of the output directory, default is TB2J_results
```

3.5 Computing Magnetocrystalline anisotropy energy (MAE) .

:warning: This feature is currently under development and internal test. Do not use it for production yet. :warning: This feature is only available with the ABACUS code.

To compute the magnetocrystalline anisotropy energy (MAE) of a magnetic system with the magnetic force theorem, two steps of DFT calculations are needed.

- The first step is to do an collinear spin calculation. The density and the Hamiltonian is saved at this step. Note that the current implementation requires the SOC to be turned on in ABACUS, but setting the SOC strength to zero (soc_lambda=0).
- The second step is to do a non-SCF non-collinear spin calculation with SOC turned on. The density is read from the previous step. In practice, one step of SCF calculation is done (as the current implementation does not write the Hamiltonian and the energy). The Hamiltonian should be saved in this step, too.

Here is one example: Step one: collinear spin calculation. Note that instead of using nspin=2, we use nspin=4, and lspinorb=1 to enable the SOC but set the soc_lambda to 0.0 to turn off the SOC. This is to make the Hamiltonian saved in the spinor form, so it can be easily compared with the next step of a real calculation with SOC.

```
INPUT_PARAMETERS
# SCF calculation with SOC turned on, but soc_lambda=0.
calculation                             scf
nspin                                   4
symmetry                                0
noncolin                                1
lspinorb                                1
ecutwfc                                 100
scf_thr                                 1e-06
init_chg                                atomic
out_mul                                 1
```

(continues on next page)

(continued from previous page)

```
out_chg                                 1
out_dos                                 0
out_band                                0
out_wfc_lcao                            1
out_mat_hs2                             1
ks_solver                               scalapack_gvx
scf_nmax                                500
out_bandgap                             0
basis_type                              lcao
gamma_only                              0
smearing_method                         gaussian
smearing_sigma                          0.01
mixing_type                             broyden
mixing_beta                             0.5
soc_lambda                              0.0
ntype                                   1
dft_functional                          PBE
```

- Step two: non-SCF non-collinear spin calculation.

In this step, we need to start from the density saved in the previous step. So we can copy the output directory of the previous step to a the present directory. To mimic the non-SCF calculation: * The scf_nmax should be set to 1 to end the scf calculation in one step. * The "scf_thr" should be set to a large value to make the calculation "converge" in one step. * The mixing_beta should be set to a small value to suppress the density mixing. * Then we can set the "init_chg" to "file" to read the density from the previous step. The lspionorb should be set to 1, and the soc_lambda should be set to a 1.0 to enable the SOC. The "out_mat_hs2" should be set to 1 to save the Hamiltonian.

:warning: Once ABACUS can output the Hamiltonian in the non-SCF calculation, we can use a "calculation=nscf" to do the non-SCF calculation.

```
INPUT_PARAMETERS
# Non-SCF non-collinear spin calculation with SOC turned on. soc_lambda=1.0
calculation                             scf
nspin                                   4
symmetry                                0
noncolin                                1
lspinorb                                1
ecutwfc                                 100
scf_thr                                 1000000.0
init_chg                                file
out_mul                                 1
out_chg                                 0
out_mat_hs2                             1
ks_solver                               scalapack_gvx
scf_nmax                                1
basis_type                              lcao
smearing_method                         gaussian
smearing_sigma                          0.01
mixing_type                             broyden
mixing_beta                             1e-06
soc_lambda                              1.0
init_wfc                                file
ntype                                   1
```

(continues on next page)

(continued from previous page)

```
dft_functional                          PBE
```

After the two steps of calculations, we can use the "abacus_get_MAE" function to compute the MAE. Essentially, the function reads the Hamiltonian from the two steps of calculations, and substract them to the the SOC part and the non-soc part. Then the non-SOC part is rotated along the different directions, and the energy difference is computed. We can define the magnetic moment directions by list of thetas/psis in the spherical coordinates, and explore the energies.

Here is an example of the usage of the function.

```
import numpy as np
from TB2J.MAE import abacus_get_MAE

def run():
    # theta, psi: along the xz plane, rotating from z to x.
    thetas = np.linspace(0, 180, 19) * np.pi / 180
    psis = np.zeros(19)
    abacus_get_MAE(
        path_nosoc= 'soc0/OUT.ABACUS',
        path_soc= 'soc1/OUT.ABACUS',
        kmesh=[6,6,1],
        gamma=True,
        thetas=thetas,
        psis=psis,
        nel = 16,
        )
                                                                    )

if __name__ == '__main__':
    run()
```

Here the soc0 and soc1 directories are where the two ABACUS calculations are done. We use a 6x6x1 Gamma-centered k-mesh for the integration (which is too small for practical usage!). And explore the energy with the magnetic moments in the x-z plane. After running the python script aboves, a file named "MAE.txt" will be created, including the theta, psi, and the energies (in eV).

3.6 Parameters in calculation of magnetic interaction parameters

3.6.1 List of parameters:

The list of parameter can be found using:

```
wann2J.py --help
```

or

```
siesta2J.py --help
```

The parameter will be explained in the following text.

- kmesh: Three integers to specify the size of a Monkhorst-Pack mesh. This is the mesh of k-points used to calculate the Green's functions. The real space supercell in which the magnetic interactions are calculated, has

the same size as the k-mesh. For example, a $7 \times 7\times$ k-mesh is linked with a $7 \times 7\times$ supercell, the atom i resides in the center cell, whereas the j atom can be in all the cells in the supercell.

- efermi: The Fermi energy in eV. For insulators, it can be inside the gap. For metals, it should be the same as in the DFT calculation. Due to the different algorithms in the integration of the density, the Fermi energy could be slightly shifted from the DFT value.
- emin, emax, and nz: During the calculation, there is a integration $\int_{emin}^{emax} d\epsilon$ calculation. The emin and emax are relative values to the Fermi energy. The emax should be 0. The emin should be low enough so that all the electronic states that affect the magnetic interactions are integrated. This can be checked with the local density of the states. Below the emin, the spin up and down density of states should be almost identical. The nz is the number of steps in this integration. The emax can be used to adjust the integration if we want to simulate the effect of the charge doping. This is with the approximation that there is only a rigid shift of the band structure. However, it is recommended to dope charge within the DFT then this approximation is not needed. The emax parameter will thus be deprecated soon.
- rcut: rcut is the cutoff distance between two ion pairs between which the magnetic interaction parameters are calculated. By default, all the pairs inside the supercell defined by the kmesh
- exclude_orbs: the indeces of orbitals, whose contribution will not be counted in the magnetic interaction. It is a list of integers. The indices are zero based.

3.7 Averaging multiple parameters

When the spins of sites i and j are along the directions $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{m}}_j$, respectively, the components of $\mathbf{J}_{ij}^{ani}$ and $\mathbf{D}_{ij}$ along those directions will be unphysical. In other words, if $\hat{\mathbf{u}}$ is a unit vector orthogonal to both $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{m}}_j$, we can only obtain the projections $\hat{\mathbf{u}}^T\mathbf{J}_{ij}^{ani}\hat{\mathbf{u}}$ and $\hat{\mathbf{u}}^T\mathbf{D}_{ij}\hat{\mathbf{u}}$. To obtain the other components, we need to rotate the spins or alternatively, rotate the structure while keeping the spin directions fixed. This method takes the approximation that the electronic strucuture is only slightly affected by the rotation of the spins, which will only lead to ne gligible relative differences in the magnetic interaction parameters. This is a good approximation for systems with weak SOC (i.e. the SOC is much weaker than the exchange-correlation). But it can fail for systems with strong SOC.

Notice that for collinear systems, there will be two orthonormal vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ that are also orthogonal to $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{m}}_j$.

The projection for $\mathbf{J}_{ij}^{ani}$ can be written as

$$\hat{\mathbf{u}}^T\mathbf{J}_{ij}^{ani}\hat{\mathbf{u}} = \hat{J}_{ij}^{xx}u_x^2 + \hat{J}_{ij}^{yy}u_y^2 + \hat{J}_{ij}^{zz}u_z^2 + 2\hat{J}_{ij}^{xy}u_xu_y + 2\hat{J}_{ij}^{yz}u_yu_z + 2\hat{J}_{ij}^{zx}u_zu_x,$$

where we considered $\mathbf{J}_{ij}^{ani}$ to be symmetric. This equation gives us a way of reconstructing $\mathbf{J}_{ij}^{ani}$ by performing TB2J calculations on rotated spin configurations. If we perform six calculations such that $\hat{\mathbf{u}}$ lies along six different directions, we obtain six linear equations that can be solved for the six independent components of $\mathbf{J}_{ij}^{ani}$. We can also reconstruct the $\mathbf{D}_{ij}$ tensor in a similar way. Moreover, if the system is collinear then only three different calculations are needed. Note that when the system is only slightly noncollinear, e.g. AFM systems with weak-ferromagnetism due to spin canting, we can still treat it as collinear and three calculations is still enough.

While rotating the spins can be done in the DFT calculation, the feature is sometimes not available for some DFT codes. In this case, an alternative is to rotate the structures while keeping the spins fixed. To account for this, TB2J provides scripts to rotate the structure named TB2J_rotate.py. The TB2J_rotate.py reads the structue file and generates three(six) files containing the rotated structures whenever the system is collinear (non-collinear). The –noncollinear parameters is used to specify whether the system is noncollinear. The output files are named atoms_i (i = 0, …, 5), where atoms_0 contains the unrotated structure. A large number of file formats is supported thanks to the ASE library and the output structure files format is provided through the –ftype parameter. An example for using the rotate file with a collinear system is:

```
TB2J_rotate.py BiFeO3.vasp --ftype vasp
```

If te system is noncollinear, then we run the following instead:

```
TB2J_rotate.py BiFeO3.vasp --ftype vasp --noncollinear
```

The user has to perform DFT single point energy calculations for the same structure with different spin orientations, or the generated structures in different directories, keeping the spins along the z direction, and run TB2J on each of them.

After producing the TB2J results for the rotated structures, we can merge the results with the following command by providing the paths to the TB2J results of the three cases:

```
TB2J_merge.py BiFeO3_1 BiFeO3_2 BiFeO3_0
```

Here the last directory will be taken as the reference structure. Note that the whole structure are rotated w.r.t. the laboratory axis but not to the cell axis. Therefore, the k-points should not be changed in both the DFT calculation and the TB2J calculation.

A new TB2J_results directory is then made which contains the merged final results.

Another method is to do the DFT calculation with spins rotated globally. That is they are rotated with respect to an axis, but their relative orientations remain the same. This can be specified in the initial magnetic moments from a DFT calculation. For calculations done with SIESTA, there is a script that rotates the density matrix file along different directions. We can then use these density matrix files to run single point calculations to obtain the required rotated magnetic configurations. An example is:

```
TB2J_rotateDM.py --fdf_fname /location/of/the/siesta/*.fdf/file
```

As in the previous case, we can use the –noncollinear parameter to generate more configurations. The merging process is performed in the same way.

3.8 The ligand spin problem: downfolding the Heisenberg Hamiltonian

Usually the magnetic moments are dominantly on the metal sites. But in some structures, the spin magnetic momentums on the ligand are non-negligible. They are however, not due to the spin splitting of the ligand atom, but the hybridization to the orbitals of surrounding atoms (often transitional metals). Therefore, the ligand magnetic moments are not independent, and will move together with the that of the metal.

In these cases, a “downfolding” method could be used to used to get an effective Heisenberg model with only the transitional metal spins as independent variables from the exchange parameters with both transitional metal spins and ligand spins. Note that the current method is only valid for ferromagnetic structures. The extenstion to antiferromagnetic and non-collinear magnetic structures are under development. Even for FM state, this result of this method should still be carefully validated. We don’t recommend the usage unless you’re sure about what you’re doing.

3.8.1 Usage:

To do the downfolding, one has to first generate the Heisenberg Hamiltonian with both the transitional metal and the ligands as magnetic elements, e.g. in CrI3,

```
wann2J.py --elements Cr I ....
```

The result is saved to TB2J_results directory. Then run the downfolding to get the Cr only effective exchange parameters, e.g.

```
TB2J_downfold.py --inpath TB2J_results --outpath TB2J_results_downfold --metals Cr --
↪ligands I
```

will generate the downfolded result to TB2J_results_downfold.

3.9 Decompose the exchange into orbital contributions.

The exchange J between two atoms can be decomposed into the sum of all the pairs of orbitals. For two atoms with m and n orbitals respectively, the decomposition can be written as m by n matrix.

Here is an example:

Since version 0.6.4, the name of the orbitals are written in the Orbital contribution section. With the Wannier90 input, the names of these orbitals are not known, thus are named as orb_1, orb_2, etc. One can search the Wannier initial projectors to see what are these orbitals.

Here is an example of cubic SrMnO3,

```
=============================================================================================
Orbitals used in decomposition:
The name of the orbitals for the decomposition:
Mn1 : ['orb_1', 'orb_2', 'orb_3', 'orb_4', 'orb_5']


=============================================================================================
Exchange:
    i      j          R        J_iso(meV)          vector          distance(A)
---------------------------------------------------------------------------------------------
   Mn1   Mn1   (  0,   0,   1) -7.1027   ( 0.000,  0.000,  3.810)  3.810
J_iso: -7.1027
Orbital contributions:
 [[ 3.184 -0.    -0.     0.    -0.   ]
 [-0.    -5.06   0.    -0.     0.   ]
 [-0.     0.    -5.06  -0.     0.   ]
 [ 0.    -0.    -0.    -0.121 -0.   ]
 [-0.     0.     0.     0.    -0.047]]
```

One can see that the contribution from the (orb_1, orb_1) pair is 3.184, and that from (orb_2, orb_2) is -5.06. Searching the wannier90 input, we can find that the orb1 and orb_2 are the dz2 and dxz orbitals, respectively.

For Siesta input, the names are known, and printed in the "orbitals use in decomposition" section. They are not the exactly the name of the siesta basis (like 3dxyZ1). For example, with a double-zeta basis set, the a 3dxy orbital might be splitted into 3dxyZ1 and 3dxyZ2. The contribution of them are summed up to make it more concise. Often, the contribution from some orbitals are negligible. In the siesta2J.py command, it's possible to specify the orbitals to be considered in the decompostion. For example, if only the 3dxy contribution of Cr is needed, one can write

```
siesta2J.py --elements Fe_3d  ....
```

If both the 3d and 4s are to be considered, one can write:

```
siesta2J.py --elements Fe_3d_4s  ....
```

For example, the bcc Fe, when only the 3d orbitals are turned on, we get:

```
================================================================================================
Orbitals used in decomposition:
The name of the orbitals for the decomposition:
Fe1 : ('3dxy', '3dyz', '3dz2', '3dxz', '3dx2-y2')

================================================================================================
Exchange:
    i      j          R        J_iso(meV)          vector          distance(A)
------------------------------------------------------------------------------------------------
   Fe1   Fe1   ( -1,   0,   0) 17.6873   (-2.467,  0.000,  0.000)  2.467
J_iso: 17.6873
Orbital contributions:
 [[11.462 -0.     0.297 -0.     0.096]
 [-0.     3.69  -0.    -0.214 -0.   ]
 [-0.163 -0.     3.215 -0.    -3.262]
 [-0.     0.396 -0.    11.451 -0.   ]
 [-0.055  0.    -3.263  0.    -4.248]]
```

where we can find in the x direction, the dxy and dxz orbitals contribute mostly to the ferromagnetic interaction, whereas the 3dx2-y2 contribution is antiferromagnetic.

Note that the option for selection of orbitals is not available in the Wannier90 interface, as the informations for labelling the orbitals are not included in the Wannier90 output (sometimes it is even not possible to do so as the Wannier functions are not necessarily atomic-orbital like).

3.9.1 Decomposition in non-collinear mode

In collinear mode, the orbital decomposition of the isotropic exchange, DMI, and anisotropic exchange is turned off by default as there are a lot of terms and boast the size of the output files for large systems. It could be turned on with the –orb_decomposition option. The orbital decomposition will be written into another file called exchange_orb_decomposition.txt. In this file, the orbital decompositions for the isotropic exchange, the three DMI vector elements (Dx, Dy, Dz) and the nine anisotropic exchange matrix elements, will be outputed as m by n matrices for each element.

3.10 The output of TB2J

In the following we describe the output files which TB2J produces. By running wann2J.py or siesta2J.py, a directory with the name TB2J_results (or the directory specified by the –output_path) will be generated, which contains the following output files:

- exchange.out: A human readable output file, which summarizes the results.
- Multibinit: A directory containing output which can be read directly by the Multibinit code.
- The exchange.out file contains three sections: cell, atoms and exchange. The cell section contains the lattice parameter matrix. The atoms section contains the positions, charges (for verification) and magnetic moments of the atoms: see example below

```
================================================================================================
Information:
Exchange parameters generated by TB2J 0.2.5.
================================================================================================
```

(continues on next page)

(continued from previous page)

```
Cell (Angstrom):
0.030   3.950   3.950
3.950   0.030   3.950
3.950   3.950   0.030

================================================================================================
Atoms:
(Note: charge and magmoms only count the wannier functions.)
Atom_number       x          y          z      w_charge    M(x)      M(y)      M(z)
Bi1             0.2413    0.2413    0.2413     2.1538   -0.0015    0.0000   -0.0044
Bi2             4.2060    4.2060    4.2060     2.1538    0.0000    0.0000    0.0044
Fe1             2.0165    2.0165    2.0165     6.3564   -0.0260    0.0000    3.7927
Fe2             5.9812    5.9812    5.9812     6.3564   -0.0176    0.0000   -3.7927
O1              5.5238    2.1558    3.9388     4.8306   -0.0013    0.0000   -0.0705
.....
Total                                          46.0038   -0.0493    0.0000   -0.0000

================================================================================================
Exchange:
i           j           R       J_iso(meV)       vector         distance(A)
------------------------------------------------------------------------------------
↪----
Fe2         Fe1   (0,   1,   1)  -28.9371 ( 3.934,  0.015,  0.015)  3.934
J_iso: -28.9371
[Testing!] Jprime: -59.620,  B: -15.342
[Testing!] DMI: (-0.5191  1.3581  0.1090)
[Testing!] J_ani:
[[ 0.    -0.002  0.047]
[-0.002  0.    -0.154]
[ 0.047 -0.154  0.   ]]
```

Here, the charge and magnetic moment of each atom are only integrated with the WFs attached to this atom. As such they can differ from the quantities coming from the direct DFT output, as not all bands are used in the construction of WFs. The WF charges should be integers, for LCAO the values depend on the band energy cutoffs. In addition, the exclusion of very deep lying levels from the calculation of J can also lead to deviations in the charges which might appear both for WFs and for LCAO. Another source of difference between the TB2J charges and magnetic moments and the DFT ones is the integration volume around the atoms, which is not necessarily the same. However, for localized d and f orbitals the magnetic moments should be close to their DFT counterparts, for TB2J to yield correct results for the parameters. Large differences between the TB2J and DFT values indicate that something may have gone wrong: either in in the contour integration $\int^{E_F} d\epsilon$ used in TB2J, or in the construction of the Wannier functions (incorrect wannierization process or too small WF basis set). Often, it comes from excluding an orbital that is important for the magnetic interaction in the studied system. In the case of metallic system, the Fermi energy might have to be slightly shifted with respect to the DFT reference due to different numerical method used in the integration of charge density.

Each pair of atoms is labeled by three parameters, the index i, j and R, where i and j are the indices in the unit cell. The vector $\vec{R}$ specifies the cell the atom j is translated to, i.e. the reduced positions of the two atoms are $\vec{r}_i$ and $\vec{r}_j + \vec{R}$, respectively. By default, the interaction is calculated within a supercell corresponding to the k-mesh. For example, with a $7 \times 7 \times 7$ k-mesh, all ij pairs will be produced for the spin labeled i in the center cell of a $7 \times 7 \times 7$ supercell. With the rcut flag, only the parameters for ij pairs within a distance of rcut are calculated. The exchange parameters are reported as follows: magnetic atom i connected with magnetic atom j, R is the lattice vector between the unit cells containing i and j, the value of J for this pair of magnetic atoms in meV, the vector connecting them, and the distance between the pair of atoms. If SOC is enabled, the DMI and anisotropic J^{ani} parameters are given in addition. The DMI vectors $\vec{D}$ and the anisotropic J^{ani} are printed as vectors and matrices, respectively.

Apart from the main exchange.out file, TB2J delivers several other outputs, which provide the input for spin dynamics (SD) and Monte Carlo (MC) simulations. TB2J is interfaced with several SD and MC codes. It has native support to the Multibinit code delivered as part of the Abinit code since version 9.0 . The TB2J_results/Multibinit directory contains the templates of input files for this code. One can usually run spin-dynamics with slight or no modification of these files. "Testing" inputs are also generated for Vampire and Thomas Ostler's GPU-ASD code.

CHAPTER

FOUR

APPLICATIONS

4.1 Magnon band structure from TB2J output

In this section we show an application of calculating the magnon band structure using the TB2J results.

There is a script within the TB2J package: TB2J_magnon.py, which can be used to plot the magnon band structure. We can show its usage by:

```
TB2J_magnon.py --help

usage: TB2J_magnon.py [-h] [--fname FNAME] [--qpath QPATH]  [--figfname FIGFNAME] [--
↪show]

TB2J_magnon: Plot magnon band structure from the TB2J magnetic interaction parameters

optional arguments:
  -h, --help     show this help message and exit
  --fname FNAME  exchange xml file name. default: exchange.xml
  --qpath QPATH  The names of special q-points. If not given, the path will be␣
↪automatically choosen. See https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.html for the␣
↪table of special kpoints and the default path.
  --figfname FIGFNAME  The file name of the figure. It should be e.g. png, pdf or other␣
↪types of files which could be generated by matplotlib.
  --show         whether to show magnon band structure.
```

The input file specified to the –fname parameter is by default exchange.xml file, which is the output in the Multibinit xml format, as can be found in the TB2J_results/Multibinit directory.

Here we take the BCC Fe as an example. After we have the TB2J_results directory, we can go to the Multibinit directory and find the exchange.xml file. The q-path can be generated automatically from the atomic structure information, using the [ASE][https://wiki.fysik.dtu.dk/ase/ase] library. We need to specify a path of q-points if the default is not what we want. The default k-path and the labels of the special kpoints can be found at this page. For example, we want a path of "Gamma-N-P-Gamma-H-N" instead of the default, it can be given by:

```
TB2J_magnon.py --qpath GNPGHN --figfname magnon.png --show
```

The magnon band structure is then written to a By using the –show parameter, the band structure is shown on screen.

From version v0.7.5, the infomation for plotting the band structure is written into a json file(magnon_band.json), together with the script for parsing the file and plot the band structure (plot_magnon_from_json_file.py).

From version v0.7.7, there is a script to plot the magnon density of states.

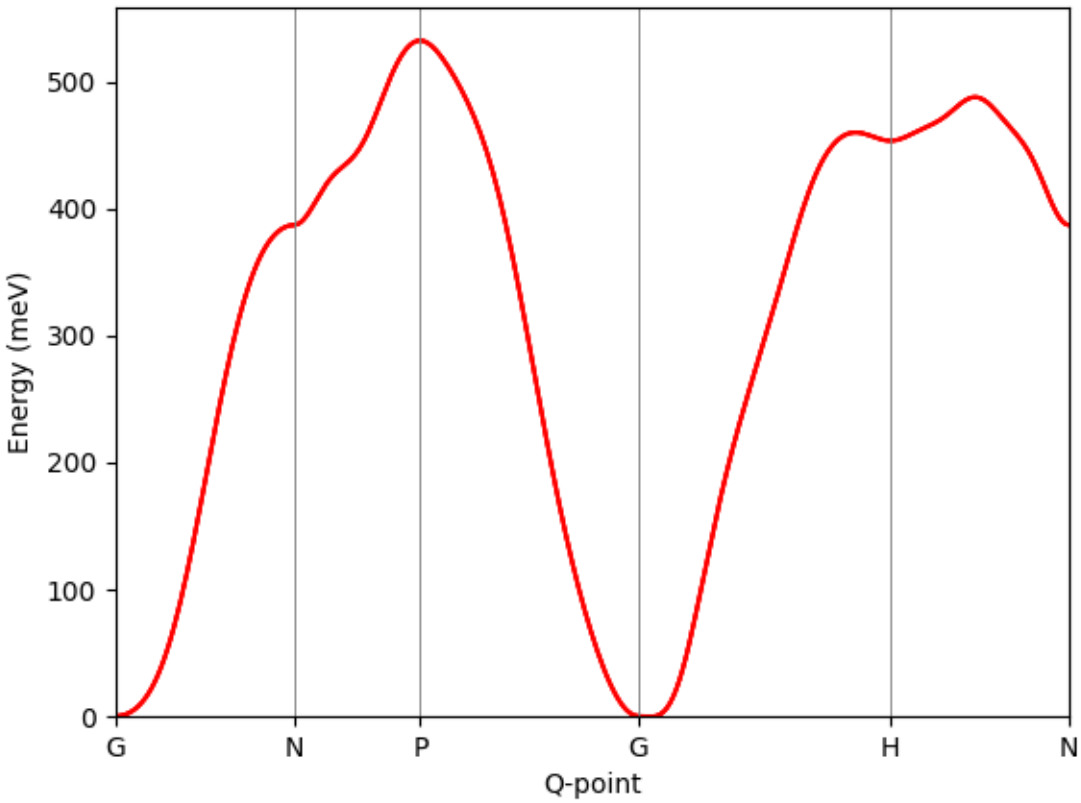


Fig. 1: exchange_magnon

```
TB2J_magnon_dos.py --help
usage: TB2J_magnon_dos.py [-h] [-p PATH] [-n NPOINTS] [-w WINDOW WINDOW] [-k KMESH KMESH␣
↪KMESH]
                          [-s SMEARING_WIDTH] [-g] [-Jq] [--show] [-f FIG_FILENAME] [-t␣
↪TXT_FILENAME]

Plot magnon DOS

options:
  -h, --help            show this help message and exit
  -p PATH, --path PATH  path to exchange.xml
  -n NPOINTS, --npoints NPOINTS
                        number of points in the energy
  -w WINDOW WINDOW, --window WINDOW WINDOW
                        energy window for the dos, two numbers giving the lower and␣
↪upper bound
  -k KMESH KMESH KMESH, --kmesh KMESH KMESH KMESH
                        k mesh
  -s SMEARING_WIDTH, --smearing_width SMEARING_WIDTH
                        Gauss smearing width in meV. Default is 10 meV.
  -g, --gamma           gamma centered k mesh
  -Jq                   use Jq
  --show                show the figure
  -f FIG_FILENAME, --fig_filename FIG_FILENAME
                        output filename for figure.
  -t TXT_FILENAME, --txt_filename TXT_FILENAME
                        output filename of the data for the magnon DOS. Default: magnon_
↪dos.txt
```

For example, with the following command, we get the DOS for bcc Fe.

```
TB2J_magnon_dos.py --show -s 10 -k 25 25 25 -f magnon_dos.png --show
```

The energies and the DOS are also saved to a txt file specified. The two columns of the file are the energies, and the DOS, respectively. It can be used for plotting the DOS if you want to plot in your own style.

4.2 Application: Spin Dynamics

The output of TB2J can be directly readed by several atomistic spin dynamics and Monte Carlo code. Currently TB2J can provide output for MULTIBINIT and Vampire.

4.2.1 Interface with Multibinit

Here we show how to use MULTIBINIT, a second principles code which can do spin dynamics. It is a part of the ABINIT package since version 9.0. The tutorial of the spin dynamics can be found on MULTIBINIT tutorial page.

Here we briefly describe how to run spin dynamics with MULTIBINIT from the TB2J outputs to with an example of BiFeO3. The example files can be found from the examples/Siesta/BiFeO3 directory.

In the TB2J_results/Mutlbinit directory, there are three files: exchange.xml, mb.in, and mb.files file. The exchange.xml file contains the Heisenberg parameters, the mb.in file is an input to the MULTIBINIT code, in which the parameters for the spin dynamics are given:

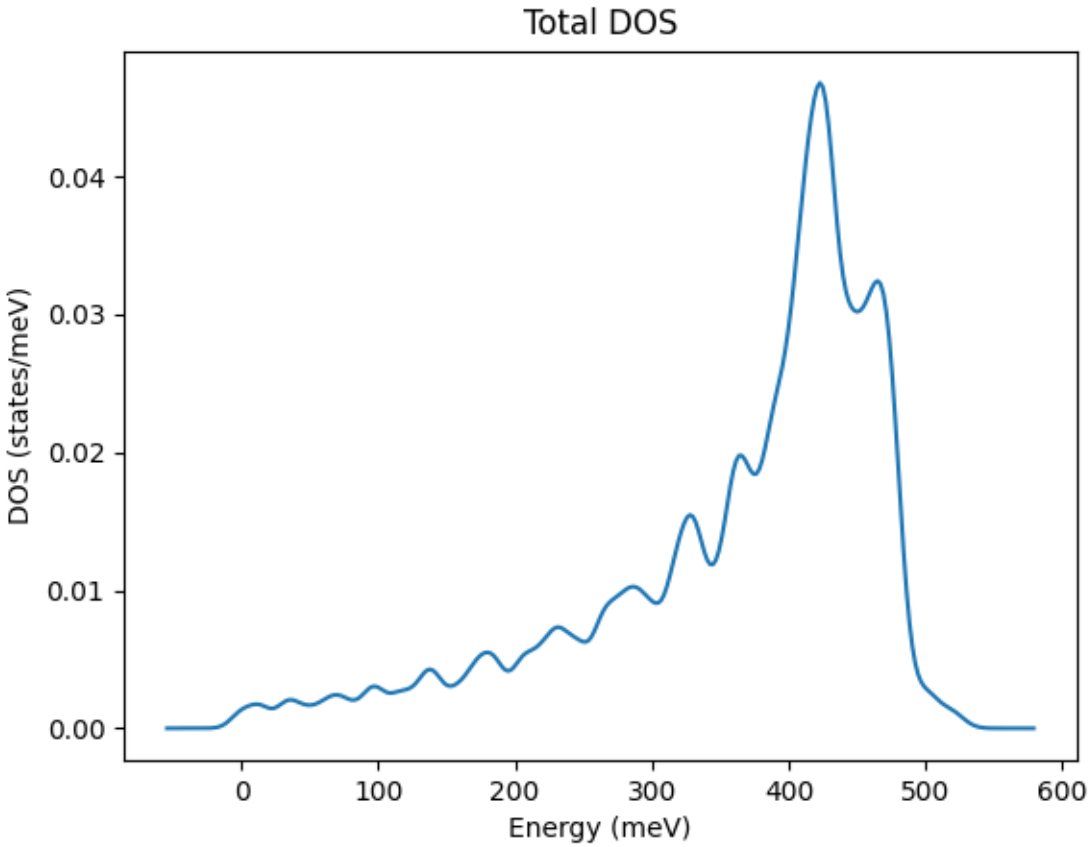


Fig. 2: magnon_dos

```
prt_model = 0

#-----------------------------------------------------------------
#Monte carlo / molecular dynamics
#-----------------------------------------------------------------
dynamics =  0    ! disable molecular dynamics

ncell =   28 28 28  ! size of supercell.
#-----------------------------------------------------------------
#Spin dynamics
#-----------------------------------------------------------------
spin_dynamics=1  ! enable spin dynamics
spin_mag_field= 0.0 0.0 0.0   ! external magnetic field
spin_ntime_pre = 10000          ! warming up steps.
spin_ntime =10000             ! number of steps.
spin_nctime=100               ! number of time steps between two nc file write
spin_dt=1e-15 s               ! time step.
spin_init_state = 1           ! FM initial state. May cause some trouble

spin_temperature=0.0

spin_var_temperature=1        ! switch on variable temperature calculation
spin_temperature_start=0      ! starting point of temperature
spin_temperature_end=1300      ! ending point of temperature.
spin_temperature_nstep= 52     ! number of temperature steps.

spin_sia_add = 1              ! add a single ion anistropy (SIA) term?
spin_sia_k1amp = 1e-6         ! amplitude of SIA (in Ha), how large should be used?
spin_sia_k1dir = 0.0 0.0 1.0  ! direction of SIA

spin_calc_thermo_obs = 1      ! calculate thermodynamics related observables
```

We can modify the default supercell size (ncell) and the temperature range to do spin dynamics in a temperature from 0K to 1300K in order to calculate the Neel temperature. We can run

```
mpirun -np 4 multibinit --F03 < mb.files
```

to run the spin dynamics. A mb.out.varT file is then generated, which has the volume heat capacit C_v, magnetic susceptibility
chi, and normalized total magnetic moment. They can be plotted as function of temperature as below, from which we can find the Neel temperature.

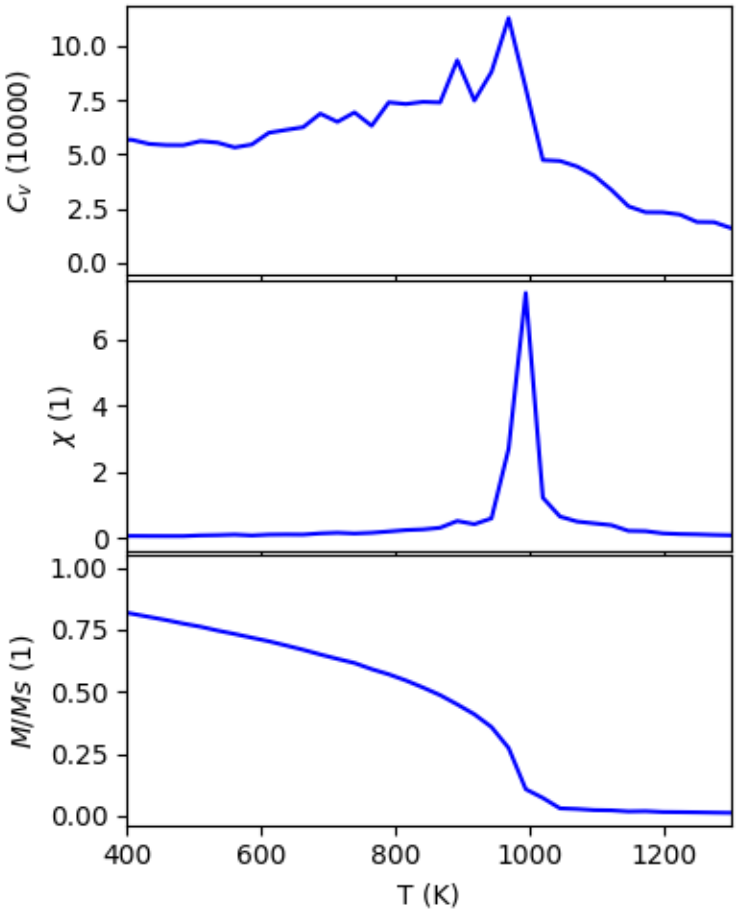
4.2.2 Interface with Vampire

Warning: there might be compatibility issues in the TB2J-Vampire interface for some versions, please contact the developpers if you encounter some of them.

A few notes about the Vampire input format:

There are 6 exchange interaction format in Vampire.

- isotropic
- vectorial


10.0
7.5
5.0
2.5
0.0
C_v (10000)
6
4
2
0
χ (1)
1.00
0.75
0.50
0.25
0.00
M/M_S (1)
400
600
800
1000
1200
T (K)

- tensorial
- normalised-isotropic
- normalised-vectorial
- normalised-tensorial

Since version 0.7.1, the TB2J-Vampire output take the "tensorial" format, in which the exchange values are 2 times that in the convention of TB2J. The anisotropic exchange and DMI are not written before version 0.7.2. They are included since then.

4.3 Writting eigen values and eigenvectors of J(q)

In this section we show how to write the eigen values and eigen vectors for a given q-point mesh. With this information, we can estimate the lowest energy spin configuration in the supercells conmensurate to the q-point mesh.

There is a script within the TB2J package: TB2J_eigen.py, which can be write the eigen value and eigen vectors. The command should be run under the TB2J_results directory.

We can show its usage by:

```
TB2J_eigen.py --help

TB2J version 0.7.1.1
Copyright (C) 2018-2020  TB2J group.
This software is distributed with the 2-Clause BSD License, without any warranty. For␣
↪more details, see the LICENSE file delivered with this software.


usage: TB2J_eigen.py [-h] [--path PATH] [--qmesh [QMESH ...]] [--gamma] [--output_fname␣
↪OUTPUT_FNAME]

TB2J_eigen.py: Write the eigen values and eigen vectors to file.

optional arguments:
  -h, --help            show this help message and exit
  --path PATH           The path of the TB2J_results file
  --qmesh [QMESH ...]   qmesh in the format of kx ky kz. Monkhorst pack or Gamma-
↪centered.
  --gamma               whether shift the qpoint grid to Gamma-centered. Default: False
  --output_fname OUTPUT_FNAME
                        The file name of the output. Default: eigenJq.txt
```

CHAPTER

FIVE

EXTENDING TB2J

In this section we show how to extend TB2J to interface with other first principles or similar codes and to write the output formats useful for codes such as spin dynamics.

5.1 Interface TB2J with other first principles or similar codes.

To interface a DFT code with TB2J, one has only to implement a tight-binding-like model which has certain methods and properties implemented. TB2J make use of the duck type feature of python, thus any class which has these things can be plugged in. Then the object can be inputted to the TB2J.Exchange class.

The methods and properties of AbstractTB class is listed as below.

class TB2J.myTB.**AbstractTB**(*R2kfactor*, *nspin*, *norb*)

HS_and_eigen(*kpts*)

get Hamiltonian, overlap matrices, eigenvalues, eigen vectors for all kpoints.

Param

- kpts: list of k points.

Returns

- H, S, eigenvalues, eigenvectors for all kpoints
- H: complex array of shape (nkpts, nbasis, nbasis)
- S: complex array of shape (nkpts, nbasis, nbasis). S=None if the basis set is orthonormal.
- evals: complex array of shape (nkpts, nbands)
- evecs: complex array of shape (nkpts, nbasis, nbands)

get_hamR(*R*)

get the Hamiltonian H(R), array of shape (nbasis, nbasis)

get_orbs()

returns the orbitals.

is_orthogonal

α used in $H(k) = \sum_R H(R) \exp(\alpha k \cdot R)$, Should be $2\pi i$ or $-2\pi i$

nbasis

nbasis=nspin*norb

norb

number of orbitals. Each orbital can have two spins.

nspin

number of spin. 1 for collinear, 2 for spinor.

xcart

The array of cartesian coordinate of all basis. shape:nbasis,3

xred

The array of cartesian coordinate of all basis. shape:nbasis,3

To pass the tight-binding-like model to the Exchange class is quite simple, here I take the sisl interface as an example

```
# read hamiltonian using sisl
fdf = sisl.get_sile(fdf_fname)
H = fdf.read_hamiltonian()
# wrap the hamiltonian to SislWrapper
tbmodel = SislWrapper(H, spin=None)
# pass to ExchangeNCL
exchange = ExchangeNCL(
    tbmodels=tbmodel,
    atoms=atoms,
    efermi=0.0,
    magnetic_elements=magnetic_elements,
    kmesh=kmesh,
    emin=emin,
    emax=emax,
    nz=nz,
    exclude_orbs=exclude_orbs,
    Rcut=Rcut,
    ne=ne,
    description=description)
exchange.run()
```

In which the SislWrapper is a AbstractTB-like class which use sisl to read the Hamiltonian and overlap matrix from Siesta output.

5.2 Extend the output to other formats

The calculated magnetic interaction parameters, together with other informations, such as the atomic structure and some metadata, are saved in "SpinIO" object. By making use of it, it is easy to output the parameters to the file format needed. Some parameters, which cannot be calculated in TB2J can also be inputted so that they can be written to the files. The list of stored data is listed below, by using which it should be easy to write the output function as a member of the SpinIO class. A method write_some_format(path) can be implemented and called in the write_all method. Then the format is automatically written after the TB2J calculation.

class TB2J.io_exchange.**SpinIO**(*atoms*, *spinat*, *charges*, *index_spin*, *orbital_names={}*, *colinear=True*, *distance_dict=None*, *exchange_Jdict=None*, *Jiso_orb=None*, *DMI_orb=None*, *Jani_orb=None*, *dJdx=None*, *dJdx2=None*, *dmi_ddict=None*, *Jani_dict=None*, *biquadratic_Jdict=None*, *debug_dict=None*, *k1=None*, *k1dir=None*, *NJT_Jdict=None*, *NJT_ddict=None*, *damping=None*, *gyro_ratio=None*, *write_experimental=True*, *description=None*)

Jani_dict

The dictionary of anisotropic exchange. The vlaues are matrices of shape (3,3).

atoms

atomic structures, ase.Atoms object

colinear

If the calculation is collinear or not

dmi_ddict

The dictionary of DMI. the key is the same as exchange_Jdict, the values are 3-d vectors (Dx, Dy, Dz).

get_DMI(*i*, *j*, *R*, *default=None*)

get_J(*i*, *j*, *R*, *default=None*)

get_J_tensor(*i*, *j*, *R*, *iso_only=False*)

Return the full exchange tensor for atom i and j, and cell R. param i : spin index i param j: spin index j param R (tuple of integers): cell index R

get_Jani(*i*, *j*, *R*, *default=None*)

Return the anisotropic exchange tensor for atom i and j, and cell R. param i : spin index i param j: spin index j param R (tuple of integers): cell index R

get_Jiso(*i*, *j*, *R*, *default=None*)

get_charge_iatom(*iatom*)

get_charge_ispin(*i*)

get_full_Jtensor_for_Rlist(*asr=False*, *iso_only=False*)

get_full_Jtensor_for_one_R(*R*)

Return the full exchange tensor of all i and j for cell R. param R (tuple of integers): cell index R returns:

Jmat: (3*nspin,3*nspin) matrix.

get_spin_iatom(*iatom*)

get_spin_ispin(*i*)

get_symbol_number_ispin(*symnum*)

Return the spin index for a given symbol number.

gyro_ratio

Gyromagnetic ratio for each atom

has_bilinear

Whether there is anisotropic exchange term

has_dmi

Whether there is DMI.

has_exchange

whether there is isotropic exchange

i_spin(*i*)

iatom(*i*)

property ind_atoms

classmethod load_pickle(*path='TB2J_results'*, *fname='TB2J.pickle'*)

model(*path*)

plot_DvsR(*ax=None*, *fname=None*, *show=False*)

plot_JanivsR(*ax=None*, *fname=None*, *show=False*)

plot_JvsR(*ax=None*, *color='blue'*, *marker='o'*, *fname=None*, *show=False*, ***kwargs*)

plot_all(*title=None*, *savefile=None*, *show=False*)

spinat

spin for each atom. shape of (natom, 3)

write_Jq(*kmesh*, *path*, *gamma=True*, *output_fname='EigenJq.txt'*, ***kwargs*)

write_all(*path='TB2J_results'*)

write_multibinit(*path*)

write_pickle(*path='TB2J_results'*, *fname='TB2J.pickle'*)

write_tom_format(*path*)

write_txt(**args*, ***kwargs*)

write_uppasd(*path*)

write_vampire(*path*)

CHAPTER

SIX

ROADMAP OF TB2J

6.1 Features to be implemented

- [ ] Magnon band structure for non-FM structures.
- [ ] Downfolding of ligand-contribution in non-collinear case.
- [ ] Generalize the merge method.
- [ ] Spin-lattice coupling.
- [ ] Single-ion anisotropy.

CHAPTER

SEVEN

ECOSYSTEM

7.1 Input to TB2J

TB2J starts from electron tight-binding-like Hamiltonian with localized basis set. Currently, this includes the Wannier-function Hamiltonian built with Wannier90, and the pseudo-atomic-orbital (PAO) based codes (SIESTA and OpenMX).

- WANNIER90: Wannier90 is an open-source code (released under GPLv2) for generating maximally-localized Wannier functions and using them to compute advanced electronic properties of materials with high efficiency and accuracy. Many electronic structure codes have an interface to Wannier90, including Quantum ESPRESSO, Abinit, VASP, Siesta, Wien2k, Fleur, OpenMX and GPAW.
- SIESTA: SIESTA is both a method and its computer program implementation, to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids. SIESTA's efficiency stems from the use of a basis set of strictly-localized atomic orbitals. A very important feature of the code is that its accuracy and cost can be tuned in a wide range, from quick exploratory calculations to highly accurate simulations matching the quality of other approaches, such as plane-wave methods. The parsing of the SIESTA output files is through sisl.
- OpenMX: OpenMX (Open source package for Material eXplorer) is a software package for nano-scale material simulations based on density functional theories (DFT), norm-conserving pseudopotentials, and pseudo-atomic localized basis functions. The methods and algorithms used in OpenMX and their implementation are carefully designed for the realization of large-scale *ab initio* electronic structure calculations on parallel computers based on the MPI or MPI/OpenMP hybrid parallelism. The TB2J-OpenMX interface is packaged in TB2J-OpenMX under the GPLv3 license.
- ABACUS ABACUS (Atomic-orbital Based Ab-initio Computation at UStc) is an open-source computer code package aiming for large-scale electronic-structure simulations from first principles, developed at the Key Laboratory of Quantum Information and Supercomputing Center, University of Science and Technology of China (USTC) - Computer Network and Information Center, Chinese of Academy (CNIC of CAS). ABACUS support three types of basis sets: pw, LCAO, and LCAO-in-pw. The TB2J-ABACUS interface can take the files from LCAO mode of ABACUS directly to compute the exchange parameters. The Wannier90 interface can be used with other types of basis set.

7.2 Spin dynamics code interfaced with TB2J

TB2J can provide the input files containing the parameters for Heisenberg models to be used in spin-dynamics code. Currently, TB2J is interfaced to MULTIBINIT and Vampire.

- MULTIBINIT: MULTIBINIT is a framework for the "second-principles" method. It is deployed in the ABINIT package. It aims at automatic mapping first-principles model to effective models which reproduce the first-principles precision but with much lower computational cost. Dynamics with multiple degrees of freedom, including lattice distortion, spin, and electron can be included in the model. The spin part of MULTIBINIT implements the atomistic spin dynamics from Heisenberg model and Landau-Lifshitz-Gilbert equations. TB2J was initially built to provide the parameters for spin model in MULTIBINIT. The documenation of spin dynamics can be found here.
- Vampire:Vampire is a high performance general purpose code for the atomistic simulation of magnetic materials. Using a variety of common simulation methods it can calculate the equilibrium and dynamic magnetic properties of a wide variety of magnetic materials and phenomena, including ferro, ferri and antiferromagnets, core-shell nanoparticles, ultrafast spin dynamics, magnetic recording media, heat assisted magnetic recording, exchange bias, magnetic multilayer films and complete devices.

7.3 Workflows

- AiiDA_TB2J_plugin: AiiDA_TB2J_plugin is a AiiDA plugin for high-throughput Siesta-TB2J calculations within the framework of AiiDA.

7.4 Codes for Linear Spin Wave method and magnon band structure

- RAD-tools: RAD-tools is a python package for the spin Hamiltonian analysis (with built-in notation changes) and magnon band structure calculation. It is interfaced directly with the TB2J .txt output ("exchange.out") and can compute the magnon band structure via the linear spin wave theory for **ferromagnetic**, **antiferromagnetic** and **spiral** magnetic structures. Documentation of the usage can be found on the package website: if you know python - use as library or if you do not know python - use console interface.

7.5 Related software without already-built interface with TB2J

There are many other tools which can be used together with TB2J, but the interface is not yet built (or made publicly available).

- UppASD: The UppASD package is a simulation tool for atomistic spin dynamics at finite temperatures. The program evolves in time the equations of motion for atomic magnetic moments in a solid. The equations take the form of the Landau-Lifshitz-Gilbert (LLG) equation. For most of the applications done so far, the magnetic exchange parameters of a classical Heisenberg Hamiltonian have been used in ASD simulations. The parameters are extracted from ab-initio DFT codes.
- Spirit: Spirit is a modern cross-platform framework for spin dynamics. Its features includes: atomistic spin lattice Heisenberg model including also DMI and dipole-dipole, Spin Dynamics simulations obeying the Landau-Lifschitz-Gilbert equation, direct energy minimization with different solvers, Minimum Energy Path calculations for transitions between different spin configurations, using the GNEB method. It provides a python package making complex simulation workflows easy, desktop UI with powerful, live 3D visualisations and direct control of most system parameters, and Modular backends including parallelization on GPU (CUDA) and CPU (OpenMP).

- SpinW: SpinW is a MATLAB library that can plot and numerically simulate magnetic structures and excitations of given spin Hamiltonian using classical Monte Carlo simulation and linear spin wave theory.

(If you know other software that can be used together with TB2J, or if you can help with interfacing with these codes, please contact us.)

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS.

8.1 How can I ask questions or report bugs?

We recommend posting the questions to the TB2J forum, https://groups.google.com/g/tb2j . Equivalently you can send emails to tb2j@googlegroups.com . Another option is the discussion page on github: https://github.com/mailhexu/TB2J/discussions . Before doing so, please read the documents and first try to find out if things are already there.

For reporting bugs, you can also open a issue on https://github.com/mailhexu/TB2J/issues .

If you meet with a bug, please first try to upgrade to the latest version to see if it is still there. And when reporting a bug, please post the inputs, the TB2J command, and the version of TB2J being used if possible. Should these files be kept secret, try to reproduce the bug in a simple system.

It is highly recommended to sign with your real name and affiliation. We appreciate the opportunity for us to get to know the community.

Any kind of feedback will help us to make improvement. Don't hesitate to get in contact!

8.2 Is it reasonable to do the DFT calculation in a magnetic non-ground state for the calculation of the exchange parameters?

It depends on how "Heisenberg" the material is. In a ideal Heisenberg model, the exchange parameters does not depend on the orientation of the spins. But in a real material it is only an approximation. Although it is a good approximation for many materials, there could be other cases that it fails.

To do such computation can be very helpful when the magnetic ground state is unknown or difficult to compute with DFT, e.g. huge supercell could be needed to model some complex magnetic states. In these cases, the estimation of the exchange parameters could be useful for finding the ground state, or provide an estimation of the other magnetic properties.

8.3 What quantities should I look into for validating the Wannier functions?

First, compare the band structure from the DFT results and that from the Wannier Hamiltonian. Then check the Wannier centers and Wannier spread to see if they are near the atom centers and the spread is small enough. From that you can also get some limited sense on the symmetry of the Wannier functions. Another thing to check in the collinear-spin case is the Re/Im ratio of the Wannier functions, which can be found in the Wannier90 output files.

8.4 How can I improve the Wannierization?

This web page provides some nice tips on how to build high quality Wannier functions: *<https://www.wanniertools.org/tutorials/high-quality-wfs>*

8.5 How can I speedup the calculation?

TB2J can be used in parallel mode, with the –np option to specify the number of processes to be used. Note that it can only use one computer node. So if you're using a job management system like slurm or pbs, you need to explicitly specify that the resources should be allocated in a single node.

8.6 Is is possible to reduce the memory usage?

TB2J need the wave functions for all k-points so it can use a lot of memory for large systems. In parallel mode, this is more of a issue as each process will store one copy of it. However, you can use the –use-cache option so that the wave functions are saved in a shared file by all the processes.

8.7 My exchange parameters are different from the results from total energy methods. What are the possible reasons?

The exchange parameters from TB2J and those from the total energy are not completely the same so they can be different.

- **TB2J uses the magnetic force theorem and perturbation theory, which is more accurate if the spin orientation only slightly deviate from the reference state.**
  The total energy method often flip the spins to get the energies with various spin configuration, therefore is probably better at describing the interactions if the spin is more disordered.
- **The results from the total energy method depends on the model it assumes.**
  For example, for system with long-range spin ineraction, or higher order interactions, these parameters are re-normalized into the exchange parameters. Whereas TB2J does not, which makes the physically meaning more tractable.
- **The conventions should be checked when you compare the results from different sources.**
  Perhaps sometimes it is just a factor of 1/2 or whether the S is normalized to 1.

8.8 The results seems to contradict the experimental results. Why?

There are many possible reasons for the discrepancies between experimental and TB2J results. Here is a incomplete list First There are many assumption made throughout the calculations, which could be unrealistic or unsuitable for the specific material.

- **In assumptions inherited from DFT calculations.**
  The Born-Oppenheimer approximation, the mean-field approximation, the LDA/GGA/metaGGA/etc.
- **In the Heisenberg model:**
  The Heisenberg model is a oversimplified model. There could be terms which are important for the specific system but are not considered in the model. For example, the higher order exchange interactions, and the interaction between the spin and other degrees of freedom (e.g. lattice vibration, charge transfer).

- **In the magnetic force theorem (MFT):**
  - The MFT is only exact as a perturbation to the ground state, which is accurate for the related properties, eg. the magnon dispersion curve. But for properties related to large deviation from the ground state, e.g. the critical temperature, the exchange parameters from the MFT might not be a good approximation (though in many material it is surprisingly good).
  - The rigid spin rotation assumption is invalid, for example, when the spins are strongly delocalized.

8.9 Does TB2J work with 2D structures or molecules?

Yes.

CHAPTER

NINE

CONTRIBUTORS

The TB2J package is initially developped:

- Xu He (mailhexu@gmail.com)
- Nicole Helbig
- Matthieu Verstraete
- Eric Bousquet

from the Phythema and Nanomat groups at the University of Liège. Part of the code is developped by Xu He during he was at ICN2 (Institut Catala de Nanociencia i Nanotecnologia) in Barcelona, Spain.

After the first public release, the following people has made significant contribution to the code:

- Andres Tellez Mora (West Virginia University)
- Aldo Romero (West Virginia University)
- Andrey Rubakov (ICMol, University of Valencia)
- Gan Jin (University of Science and Technology of China)
- Zhenxiong Shen (University of Science and Technology of China)

We thank all the other people who have contributed to the code, including the ones who have reported bugs or typos,
suggested improvements, and asked questions.

We welcome contributions. You can help us with: - extending the input format to other codes, e.g. first principles or tight binding code. - extending the output to other spin dynamics code. - extending the algorithm so that it can be used to calculate other parameters. - testing the validity of the methods implemented. - improving the documentation.

Please also feel free to contact us if you think there are other things you can help.

CHAPTER

TEN

REFERENCES

The implementation details of TB2J:

Xu He, Nicole Helbig, Matthieu J. Verstraete, Eric Bousquet, TB2J: a python package for computing magnetic interaction parameters. Computer Physics Communications, 107938 (2021).

The theoretical background for exchanges calculation is described in

- Initial idea of Green's function method of calculating J.

  Liechtenstein et al. J.M.M.M. 67,65-74 (1987), (aka LKAG)

- Isotropic exchange using Maximally localized Wannier function.

  Korotin et al. Phys. Rev. B 91, 224405 (2015)

- Full exchange tensor.

  Antropov et al. Physica B 237-238 (1997) 336-340

- Biqudratic term.

  S. Lounis, P. H. Dederichs, Phys. Rev. B 82 180404(R) (2010)

  Szilva et al, Phys. Rev. Lett. 111, 127204

- **Downfold method**
  I. V. Solovyev, Phys. Rev. B 103, 104428

CHAPTER

ELEVEN

DEVELOPMENT

Note: This page is currently under development.

This tutorial gives a few guidelines on how to develop in TB2J.

We hope to make TB2J easily accessible to developers.

11.1 Code

11.1.1 Writting documenation

The documents are in the directory docs, and written in markdown and restructed text (rst) format.

In the docs/index.rst, the main page of the documentation and the links to the table of contents in the sidebar are defined.

In the docs/src, each topic is written in a rst or md file.

CHAPTER

TWELVE

RELEASE NOTES

12.1 v0.9.0 March 22, 2024

Improved merge method for anisotropic exchange and DMI. (thanks to Andres Tellez Mora!)

12.2 v0.8.2 March 4, 2024

TB2J can now read the "tb.dat" file instead of the "hr.dat"+"centers.xyz" files.

(>=0.8.2.2) Allow atom symbols+number format (e.g. Fe1, Fe2) in Wannier .win file, and in the –magnetic_elements option. (issue46) Allow synthetic atom in siesta (>=0.8.2.4). Print actual emin in non-collinear mode. (0.8.2.5) Reduce memory usage by not computing density matrix from Green's function. (0.8.2.6)

12.3 v0.8.1 Febrary 25, 2024

Interface with ABACUS for non-collinaer spin calculations is implemented.

12.4 v0.8.0 Febrary 18, 2024

Add a new DFT code interface to ABACUS! Thanks to Zhen-Xiong Shen and Gan Jin from the ABACUS team for providing the coding for parsing the ABACUS output files. In this version the colliear spin is implemented and the non-collinear will be soon added.

12.5 v0.7.7 October 11, 2023

Added script: TB2J_magnon_dos.py for plotting the magnon density of states. See https://tb2j.readthedocs.io/en/latest/src/magnon_band.html

12.6 v0.7.6 May 10, 2023

TB2J_magnon.py now writes the band structure infomation into a json file. A script to read the json file and plot the band structure is in the same directory.

12.7 v0.7.3.1 July 24, 2022

Improve error message for wannier functions badly localized to atomic centers.

12.8 v0.7.3 June 8, 2022

The Vampire output include the DMI and anisotropic exchange. The TB2J_downfold.py is extended to DMI and anisotropic exchange (for early test only).

12.9 v0.7.2.1 April 20, 2022

Fix compatibility issue with Python3.10

12.10 v0.7.2 March 01, 2022

Add TB2J_eigen.py script to write the eigen values and eigenvectors of the J(q) in a qpoint mesh. Remove J' and B from the output, which are often not useful and confusing.

12.11 v0.7.1 January 04, 2022

Bug fix: convention in Vampire output (tensor->tensorial, and a factor of 2 added to the exchange values). Some documentation about the Vampire format added. (Contributions from Jun Gyu Lee.)

12.12 v0.7.0 October 05, 2021

Allow to do orbital decompositions to isotropic/anisotropic exchange and DMI with --orb_decomposition.

12.13 v0.6.10 September 29, 2021

Bug fix: wrong matrix alignment in orbital decomposition for atom pairs with different species.

12.14 v0.6.9 September 15, 2021

Better xticks in magnon bands.

12.15 v0.6.8 September 15, 2021

Bug fix: downfolding with non-magnetic atoms now works. Bug fix: Error reading from user specified structural input file.

12.16 v0.6.7 September 10, 2021

Use tqdm & p_tqdm instead of progressbar. Fix progressbar in parallel mode.

12.17 v0.6.6 September 1, 2021

Output the figure of J vs distance. Fix a bug when the orbitals are not grouped by atoms.

12.18 v0.6.4 August 9, 2021

Documentation of orbital decomposition. More concise orbital decompostion to the exchange with siesta2J.py. Allow to specify the orbitals used in the decomposition in siesta2J.py --element option.

12.19 v0.6.3 July 3, 2021

Revert qsolver to old version.

12.20 v0.6.2 May 27, 2021

Enable reading structures from wannier .win file so --posfile is no more necessray.

: --posfile option can still be used.

12.21 v0.6.1

Change the internal order of orbitals in noncollinear Tight-binding.

A script for building docker image has been added (Nikolas Garofil).

12.22 v0.6.0

Add TB2J_downfold.py script to deal with ligand spin contribution.

12.23 v0.5.0

Add Wannier input from banddownfolder package using --wannier_type=banddownfolder. Currently only collinear calculation supported.

12.24 v0.4.4 March 16, 2021

Allow parallel over k in tight binding eigen solver.

12.25 v0.4.3 March 11, 2021

Add Reference to TB2J paper.

12.26 v0.4.2 March 11, 2021

Fix a bug that the atoms scaled positions get wrapped. Fix a bug with consecutive parallel run in python mode.

12.27 v0.4.1 February 2, 2021

Use a Legendre path for the integration which is more stable and requires less poles(--nz). Memory optimization.

12.28 v0.4.0 February 1, 2021

Add --np option to specify number of cpu cores in parallel. Dependency on pathos is added.

12.29 v0.3.8 December 29, 2020

Add --output_path option to specify the output path.

12.30 v0.3.6 December 7, 2020

Use Simpson's rule instead of Euler for integration.

12.31 v0.3.5 November 3, 2020

Add --groupby option in wann2J.py to specify the order of the basis set in the hamiltonian.

12.32 v0.3.3 September 12, 2020

- Use collinear exchange calculator for siesta-collinear calculation, which is faster.

12.33 v0.3.2 September 12, 2020

```
add \--use\_cache option to reduce the memory usage by storing the Hamiltonian

:   and eigenvectors on disk using memory map.
```

12.34 v0.3.1 September 3, 2020

- A bug in the sign of the magnetization along y in Wannier and OpenMX mode is fixed.

12.35 v0.3 August 31, 2020

- A bug in calculation of anisotropic exchange is fixed.
- add TB2J_merge.py for merging DMI and anisotropic exchange from calculations with different spin orientation or structure rotation.
- Improvement on output txt file.
- An interface to OpenMX (TB2J_OpenMX) is added in a separate github under GPLv3. at https://github.com/mailhexu/TB2J-OpenMX
- Many improvement and bugfixes

12.36 v0.2 2020

- Moved to github
- DMI and anisotropic exchange
- Magnon band structure (For FM and single magnetic specie)
- Siesta Input
- Documentation on readthedocs

12.37 v0.1 2018

- Initial version on gitlab.abinit.org
- Isotropic exchange
- Wannier function as input
- Interface with Multibinit, Tom's ASD, and Vampire

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex

INDEX

X