TB2J

Xu He

Jun 17, 2026

CONTENTS:

1 Installation **3**
- 1.1 Dependencies . . . 3

2 Conventions of Heisenberg Model **5**

3 Tutorial **7**
- 3.1 Use TB2J with Wannier90 . . . 7
- 3.2 Use TB2J with Siesta . . . 10
- 3.3 Use TB2J with OpenMX . . . 11
- 3.4 Use TB2J with ABACUS . . . 11
- 3.5 Computing Magnetocrystalline anisotropy energy (MAE) . . . 14
- 3.6 Parameters for Magnetic Interaction Calculations . . . 17
- 3.7 Averaging multiple parameters . . . 18
- 3.8 The ligand spin problem: downfolding the Heisenberg Hamiltonian . . . 19
- 3.9 Decompose the exchange into orbital contributions. . . . 20
- 3.10 Symmetrization of the exchange parameters . . . 22
- 3.11 The output of TB2J . . . 22

4 Applications **25**
- 4.1 Magnon Band Structure . . . 25
- 4.2 Application: Spin Dynamics . . . 26
- 4.3 Writting eigen values and eigenvectors of J(q) . . . 29

5 Magnon Band Structure and Density of States: Theory **31**
- 5.1 Introduction . . . 31
- 5.2 Heisenberg Model . . . 31
- 5.3 Fourier Transform to Reciprocal Space . . . 32
- 5.4 Magnon Hamiltonian Construction . . . 32
- 5.5 Band Structure Calculation . . . 33
- 5.6 Density of States Calculation . . . 34
- 5.7 Energy Units . . . 35
- 5.8 Validation and Quality Checks . . . 35
- 5.9 Spin Quantization . . . 35
- 5.10 References . . . 36
- 5.11 See Also . . . 36

6 SPRKKR Magnon Workflow **37**
- 6.1 Python API . . . 37
- 6.2 CLI . . . 38

7 TB2J_edit **39**

7.1 Command Line Interface . . . 39
7.2 Python API . . . 40

8 **Extending TB2J** **43**
8.1 Interface TB2J with other first principles or similar codes. . . . 43
8.2 Extend the output to other formats . . . 44

9 **Roadmap of TB2J** **45**
9.1 Features to be implemented . . . 45

10 **Ecosystem** **47**
10.1 Input to TB2J . . . 47
10.2 Spin dynamics code interfaced with TB2J . . . 47
10.3 Workflows . . . 48
10.4 Codes for Spin Wave methods . . . 48
10.5 Related software without already-built interface with TB2J . . . 48

11 **Frequently asked questions.** **49**
11.1 How can I ask questions or report bugs? . . . 49
11.2 Is it reasonable to do the DFT calculation in a magnetic non-ground state for the calculation of the exchange parameters? . . . 49
11.3 What quantities should I look into for validating the Wannier functions? . . . 49
11.4 How can I improve the Wannierization? . . . 50
11.5 How can I speedup the calculation? . . . 50
11.6 Is is possible to reduce the memory usage? . . . 50
11.7 My exchange parameters are different from the results from total energy methods. What are the possible reasons? . . . 50
11.8 The results seems to contradict the experimental results. Why? . . . 50
11.9 Does TB2J work with 2D structures or molecules? . . . 51

12 **Contributors** **53**

13 **References** **55**

14 **Development** **57**
14.1 Code . . . 57

15 **Release Notes** **59**
15.1 Current development version (v1.0.0-alpha) . . . 59
15.2 v0.11.0 October 10, 2024 . . . 59
15.3 v0.10.0 September 1, 2024 . . . 59
15.4 v0.9.0 March 22, 2024 . . . 59
15.5 v0.8.2 March 4, 2024 . . . 60
15.6 v0.8.1 Febrary 25, 2024 . . . 60
15.7 v0.8.0 Febrary 18, 2024 . . . 60
15.8 v0.7.7 October 11, 2023 . . . 60
15.9 v0.7.6 May 10, 2023 . . . 60
15.10 v0.7.3.1 July 24, 2022 . . . 60
15.11 v0.7.3 June 8, 2022 . . . 60
15.12 v0.7.2.1 April 20, 2022 . . . 60
15.13 v0.7.2 March 01, 2022 . . . 60
15.14 v0.7.1 January 04, 2022 . . . 61
15.15 v0.7.0 October 05, 2021 . . . 61
15.16 v0.6.10 September 29, 2021 . . . 61
15.17 v0.6.9 September 15, 2021 . . . 61

15.18 v0.6.8 September 15, 2021 61
15.19 v0.6.7 September 10, 2021 61
15.20 v0.6.6 September 1, 2021 61
15.21 v0.6.4 August 9, 2021 61
15.22 v0.6.3 July 3, 2021 61
15.23 v0.6.2 May 27, 2021 61
15.24 v0.6.1 62
15.25 v0.6.0 62
15.26 v0.5.0 62
15.27 v0.4.4 March 16, 2021 62
15.28 v0.4.3 March 11, 2021 62
15.29 v0.4.2 March 11, 2021 62
15.30 v0.4.1 February 2, 2021 62
15.31 v0.4.0 February 1, 2021 62
15.32 v0.3.8 December 29, 2020 62
15.33 v0.3.6 December 7, 2020 62
15.34 v0.3.5 November 3, 2020 62
15.35 v0.3.3 September 12, 2020 63
15.36 v0.3.2 September 12, 2020 63
15.37 v0.3.1 September 3, 2020 63
15.38 v0.3 August 31, 2020 63
15.39 v0.2 2020 63
15.40 v0.1 2018 63

16 Indices and tables **65**

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

The easiest 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 are permission problems.

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

You can install optional dependencies for specific interfaces or all of them:

- For the Siesta interface:

```
pip install TB2J[siesta]
```

- For the lawaf interface:

```
pip install TB2J[lawaf]
```

- To install all optional dependencies:

```
pip install TB2J[all]
```

1.1 Dependencies

TB2J is a python package which requires python version higher than 3.8 to work. It depends on the following packages.

- numpy<2.0
- scipy
- matplotlib
- ase>=3.19
- tqdm
- pathos
- packaging>=20.0
- HamiltonIO>=0.2.4
- pre-commit

- sympair>0.1.0
- tomli>=2.0.0
- tomli-w>=1.0.0

If you use pip to install, 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
- netcdf4 (optional) for Siesta interface
- lawaf==0.2.3 (optional) for lawaf interface
- GPAW (optional) For gpaw interface (not yet fully operational).

By default, TB2J only forces the non-optional dependencies to be installed automatically. The *sisl* and *netcdf4* packages, which are used to read the Hamiltonian from the Siesta output, are optional and can be installed with the *[siesta]* extra. For example:

```
pip install TB2J[siesta]
```

installed automatically. The sisl package (with netcdf4) 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 netcdf4
```

1.1.1 How to install in a virtual environment

It is recommended to install TB2J in a virtual environment (venv), which is like a world parallel to the main Python environment where other packages are installed. With this, conflicts between library versions can be avoided. For example, you may have other packages that only work with an old version of numpy, whereas TB2J requires a newer version. Within this venv, you can have the new version without worrying about conflicts.

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 by 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.

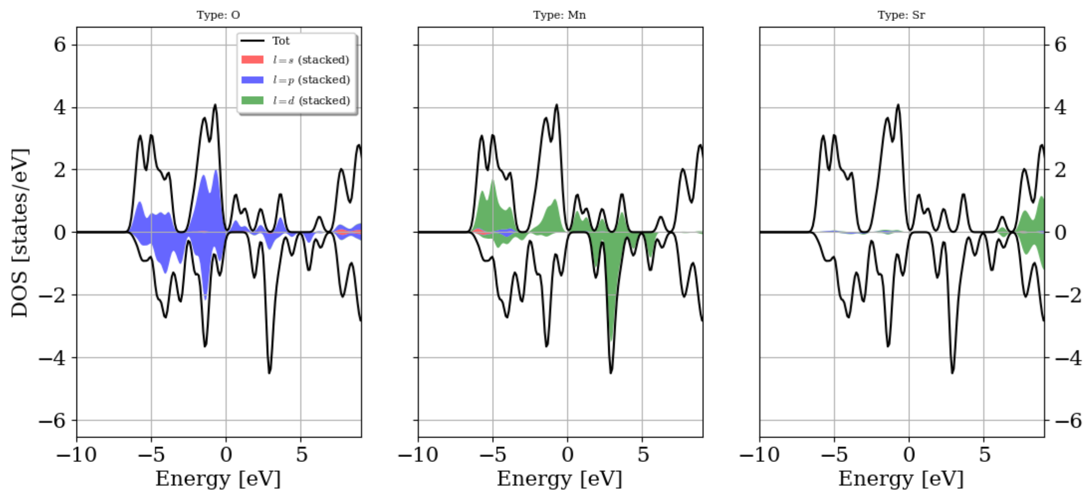
As we can see, the Mn 3d and O 2p orbitals should be included into the WF Hamiltonian. The Sr 4d orbitals are too high in energy, so we exclude them from the WF Hamiltonian.

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
```

(continues on next page)



(continued from previous page)

```
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
  --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 and SIESTA code.

The MAE calculation is implemented in `TB2J/MAEGreen.py` using the magnetic force theorem with Green's function perturbation theory. The implementation provides both efficient perturbative and exact eigenvalue approaches for computing magnetocrystalline anisotropy energies.

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
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
dft_functional                          PBE
```

After the two steps of calculations, we can use the `abacus_get_MAE` function from `TB2J.MAEGreen` to compute the MAE. The function reads the Hamiltonian from the two steps of calculations and uses the magnetic force theorem to compute the band energy differences for different magnetization directions.

The non-SOC Hamiltonian is rotated along different magnetization directions (defined by theta, phi in spherical coordinates), and the energy differences are computed using second-order perturbation theory.

Here is an example of the usage of the function.

```python
import numpy as np
from TB2J.MAEGreen import abacus_get_MAE

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

(continues on next page)

(continued from previous page)

```
        phis=phis,
        nel = 16,
        )

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

3.5.1 Advanced Usage

The `MAEGreen` class also supports various angle presets and options:

```
from TB2J.MAEGreen import MAEGreen

# Use predefined angle sets
mae = MAEGreen(
    tbmodels=model,
    atoms=model.atoms,
    kmesh=[6,6,1],
    angles="xyz",  # x, y, z axes
    # angles="fib",  # Fibonacci sphere sampling
    # angles="miller",  # Miller index directions
    # angles="scan",  # Full angular scan
)

# Custom angle arrays
mae = MAEGreen(
    tbmodels=model,
    atoms=model.atoms,
    kmesh=[6,6,1],
    angles=[thetas, phis],  # Custom theta, phi arrays
)

# Run with both perturbation and eigenvalue methods
mae.run(output_path="my_mae_results", with_eigen=True)
```

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 above, several output files will be created in the `TB2J_anisotropy` directory:

- **MAE.dat**: Contains theta, phi, and total MAE values (in meV)
- **MAE_matrix.dat**: Atom-atom interaction matrices for each angle
- **MAE_orb.dat**: Orbital-resolved MAE contributions for each atom
- **MAE_eigen.dat**: (optional) Exact eigenvalue results if `with_eigen=True`

The MAE values are computed using second-order perturbation theory and reported in meV relative to the reference magnetization direction.

3.6 Parameters for Magnetic Interaction Calculations

3.6.1 Introduction

TB2J provides several command-line tools to calculate magnetic interactions from different electronic structure codes. Each tool shares a common set of parameters for controlling the magnetic interaction calculations, while also having some tool-specific parameters.

To view the complete list of parameters for any tool, use the –help option:

```
wann2J.py --help    # For Wannier90-based calculations
siesta2J.py --help  # For SIESTA-based calculations
abacus2J.py --help  # For ABACUS-based calculations
```

3.6.2 Common Parameters

These parameters are available across all TB2J tools and control the core aspects of magnetic interaction calculations.

System Definition

- elements: List of magnetic elements to include in the Heisenberg model. Example: `--elements Fe Ni`
- index_magnetic_atoms: Explicitly specify magnetic atoms by their indices (starting from 1) in the unit cell. Overrides element-based selection. Example: `--index_magnetic_atoms 1 2 4`
- exclude_orbs: Zero-based indices of orbitals to exclude from magnetic site calculations. Useful for fine-tuning the magnetic model.

K-space and Real Space Parameters

- **kmesh: Three integers defining the Monkhorst-Pack k-point mesh (e.g., `--kmesh 7 7 7`). This mesh determines:**
  - The k-points used for Green's function calculations
  - The real-space supercell size for magnetic interactions
  - For example, a $7 \times 7 \times 7$ k-mesh creates a supercell where magnetic interactions are calculated between the central atom and all atoms within this supercell
- rcut: Maximum distance (in Å) for calculating magnetic interactions between atom pairs. If not specified, includes all pairs within the supercell defined by kmesh.

Energy Integration Parameters

- **efermi: Fermi energy in eV. Handling varies by tool:**
  - siesta2J.py/abacus2J.py: Automatically read from output
  - wann2J.py: Must be provided from DFT calculation
  - For insulators: Can be within band gap
  - For metals: Should match the DFT value
- smearing: Fermi smearing width for density computation and CFR. Controls the temperature used in numerical integration. Accepts values like "600K" or "0.2eV". Default: 600K.
- **emin, emax: Energy range for integration, relative to efermi:**
  - emin: Should be low enough to include all magnetically relevant states
  - Check local density of states: spin up/down should be nearly identical below emin

  - emax: Usually 0.0 (Fermi level). Can be adjusted for charge doping studies
  - Note: Direct DFT doping is preferred over emax adjustment
- nz: Number of energy points for numerical integration of $\int_{emin}^{emax} d\epsilon$

Performance and Output Controls

- np/nproc: Number of CPU cores for parallel processing. Default: 1
- use_cache: Store wavefunctions and Hamiltonian on disk to reduce memory usage. Useful for large systems. Default: False
- cutoff: Minimum magnitude of exchange coupling (J) to write to output (in eV). Helps filter numerical noise.
- output_path: Directory for output files. Default: TB2J_results

Output Customization

- **description: Add essential calculation details to the XML output:**
  - Exchange-correlation functional used
  - Hubbard U values if applicable
  - Magnetic state description
  - Any other relevant parameters

Advanced Options

- orb_decomposition: Analyze orbital contributions in non-collinear calculations. Default: False
- orth: Apply Löwdin orthogonalization before diagonalization. Usually for testing purposes. Default: False

3.6.3 Tool-Specific Parameters

SIESTA Interface (siesta2J.py)

- fdf_fname: Path to SIESTA's input fdf file. Default: ./
- fname: Output exchange parameters XML filename. Default: exchange.xml
- split_soc: Enable reading of spin-orbit coupling from SIESTA output. Default: False

ABACUS Interface (abacus2J.py)

- path: Location of ABACUS calculation files. Default: ./
- suffix: ABACUS calculation label (used in OUT.suffix). Default: abacus

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

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

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 Symmetrization of the exchange parameters

The exchange parameters obtained from the output of TB2J may not preserve the symmetry of the crystal. This can be attributed to two reasons:

1. The magnetic state breaks the symmetry of the crystal. As TB2J perturbs this magnetic state, the J values should reflect the symmetry of the magnetic state.
2. Numerical noise can be present in the calculation, either during the DFT stage, the Wannierization procedure, or the post-processing within TB2J. This numerical noise should ideally be very small. If it is not, the calculation procedure should be optimized.

In some cases, it is necessary for the exchange parameters to strictly adhere to the symmetry of the crystal structure. To achieve this, you can use the script `TB2J_symmetrize.py` to symmetrize the exchange parameters.

:Warning: Please note that the current version of the script only considers the symmetry of the crystal and does not take the magnetic moment into account.

:Warning: Additionally, only the isotropic exchange is symmetrized in this version. The symmetrization of the DMI and anisotropic exchange will be implemented in a future version.

The `TB2J_symmetrize.py` script utilizes the symmetry of the crystal to identify symmetrically equivalent atom pairs and the corresponding symmetry operators. It then averages the J values over these symmetrically equivalent atom pairs.

Here is the usage information for the script:

```
usage: TB2J_symmetrize.py [-h] [-i INPATH] [-o OUTPATH] [-s SYMPREC]

Symmetrize exchange parameters. Currently, it takes the crystal symmetry into account␣
↪and not the magnetic moment.
Also, only the isotropic exchange is symmetrized in this version. The symmetrization of␣
↪the DMI and anisotropic exchange will be implemented in a future version.

options:
  -h, --help            show this help message and exit
  -i INPATH, --inpath INPATH
                        input path to the exchange parameters
  -o OUTPATH, --outpath OUTPATH
                        output path to the symmetrized exchange parameters
  -s SYMPREC, --symprec SYMPREC
                        precision for symmetry detection. The default value is 1e-5␣
↪Angstrom
```

An example:

```
TB2J_symmetrize.py -i TB2J_results -o TB2J_symmetrized -s 1e-4
```

The script can read the data from TB2J_results and write the symmetrized exchange parameters to a new directory TB2J_symmetrized. The precistion for detecting the symmetry is 1e-4 angstrom.

3.11 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.
==========================================================================================
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.   ]]
Combined J tensor (meV) [J = Jiso*I + DMI_antisymmetric + Jani_symmetric]:
[[-28.937   0.107  -1.311]
[ -0.111  -28.937  -0.673]
[  1.405   0.365  -28.937]]
```

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.

For convenience and verification, TB2J also outputs the combined exchange tensor in matrix form, which is the sum of all three contributions:

$$\mathbf{J} = J^{iso}\mathbf{I} + \mathbf{J}^{DMI} + \mathbf{J}^{ani}$$

where:

- $J^{iso}\mathbf{I}$ is the isotropic exchange (diagonal matrix)
- $\mathbf{J}^{DMI}$ is the antisymmetric DMI tensor:

$$\mathbf{J}^{DMI} = \begin{pmatrix} 0 & D_z & -D_y \\ -D_z & 0 & D_x \\ D_y & -D_x & 0 \end{pmatrix}$$

- $\mathbf{J}^{ani}$ is the symmetric anisotropic exchange tensor

The combined tensor is useful for verifying the sign conventions and understanding the total exchange interaction in tensor form.

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

For the theoretical background of magnon calculations, see *Magnon Band Structure and Density of States: Theory*.

TB2J provides a unified command-line tool *TB2J_magnon.py* for magnon band structure and DOS calculations.

4.1.1 New Interface (v0.9.12+)

4.1.2 Quick Start

```
# Plot magnon band structure
TB2J_magnon.py --bands

# Plot magnon DOS
TB2J_magnon.py --dos

# Plot both
TB2J_magnon.py --bands --dos

# Exclude specific interactions
TB2J_magnon.py --bands --no-Jani --no-DMI
```

4.1.3 Legacy Interface

There are also older scripts within the TB2J package for magnon calculations:

TB2J_magnon.py (Legacy)

This script can be used to plot magnon band structure from the Multibinit XML format:

```
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␣
```

(continues on next page)

(continued from previous page)

```
↪automatically chosen.
  --figfname FIGFNAME  The file name of the figure.
  --show         whether to show magnon band structure.
```

The input file (–fname) is by default exchange.xml, which can be found in the TB2J_results/Multibinit directory.

Example usage with BCC Fe:

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

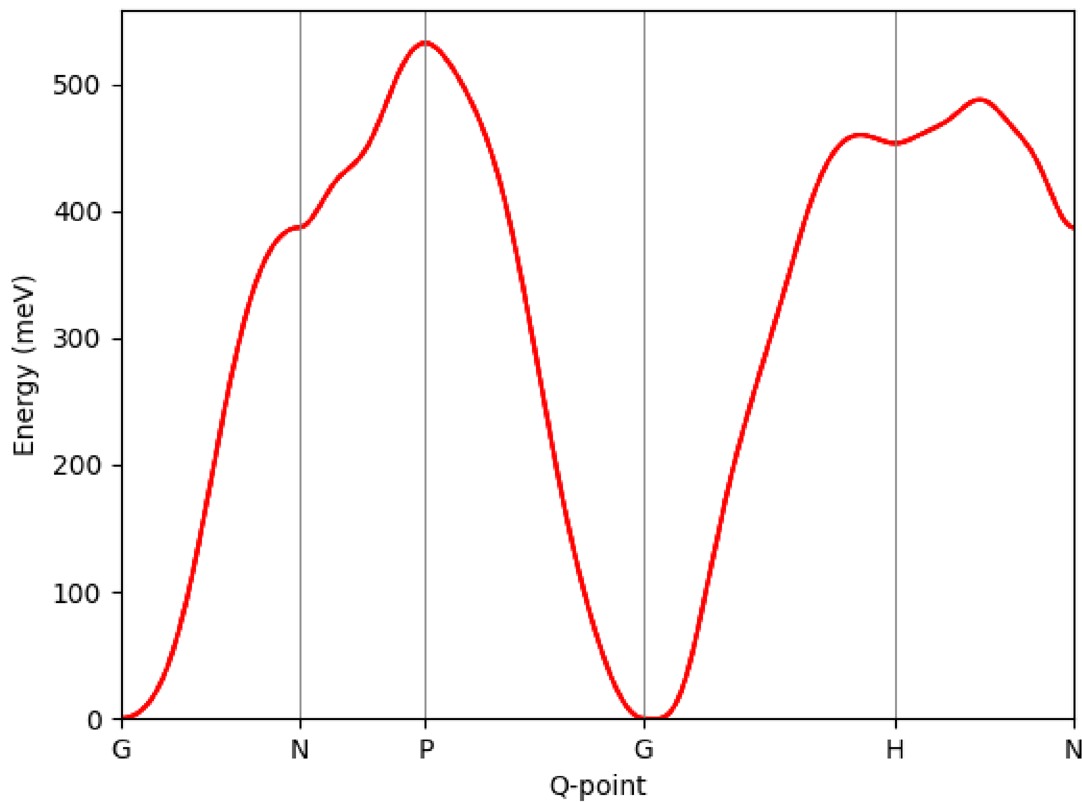


Fig. 1: exchange_magnon

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

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.

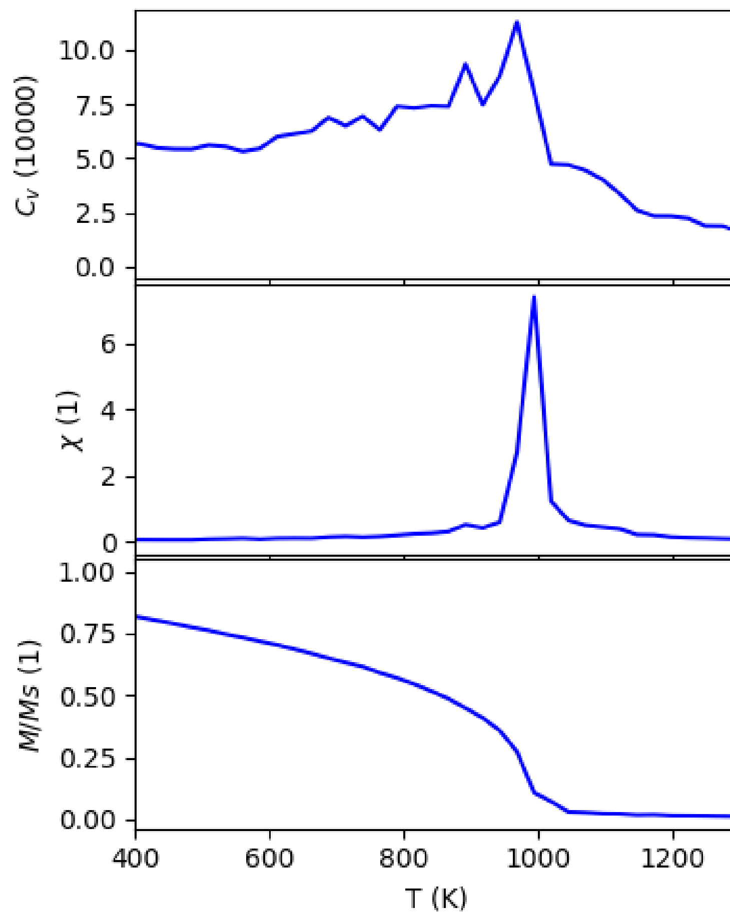
In the TB2J_results/Mutlbinit directory, there are three files: exchange.xml, mb.in, and mb.files file. The exchange.xml file contains the Heisenberg parameters, the mb.in file is an input to the MULTIBINIT code, in which the parameters for the spin dynamics are given:

```
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.



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

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
- 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 exchange.out file. This convention is automatically applied in the TB2J-Vampire interface. 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

MAGNON BAND STRUCTURE AND DENSITY OF STATES: THEORY

```
For usage instructions and command-line interface, see [Magnon Band Structure](magnon_band.rst).
```

This document describes the theoretical background for calculating magnon band structures and density of states (DOS) from exchange parameters computed by TB2J.

5.1 Introduction

Magnons are quantized spin waves in magnetic materials. Understanding magnon dispersions is essential for:

- Predicting thermodynamic properties (magnetization, specific heat)
- Analyzing magnetic stability and phase transitions
- Designing spintronic devices
- Interpreting neutron scattering experiments

TB2J calculates magnon properties from first-principles-derived exchange parameters using linear spin wave theory (LSWT).

5.2 Heisenberg Model

5.2.1 General Spin Hamiltonian

The magnetic interactions in a crystal are described by a generalized Heisenberg Hamiltonian:

$$ \mathcal{H} = -\sum_{i,j,\mathbf{R}} \mathbf{S}_i \cdot \mathbf{J}_{ij}(\mathbf{R}) \cdot \mathbf{S}_j(\mathbf{R}) $$

where:

- i, j index magnetic atoms in the unit cell
- $\mathbf{R}$ is the lattice vector connecting unit cells
- $\mathbf{S}_i$ is the spin operator at site i
- $\mathbf{J}_{ij}(\mathbf{R})$ is the 3×3 exchange tensor

5.2.2 Exchange Tensor Components

The exchange tensor can be decomposed into:

$$ \mathbf{J}{ij} = J{ij}^{\text{iso}} \mathbf{I} + \mathbf{J}{ij}^{\text{sym}} + \mathbf{D}{ij} \times $$

where:

- J_{ij}^{iso}: Isotropic (Heisenberg) exchange
- $\mathbf{J}_{ij}^{\text{sym}}$: Symmetric anisotropic exchange (traceless)
- $\mathbf{D}_{ij}$: Dzyaloshinskii-Moriya interaction (DMI) vector

In component form:

$$ J_{ij}^{\alpha\beta} = J_{ij}^{\text{iso}}\delta_{\alpha\beta} + J_{ij}^{\text{sym},\alpha\beta} + \epsilon_{\alpha\beta\gamma} D_{ij}^\gamma $$

5.2.3 Single-Ion Anisotropy

For on-site terms ($i=j$, $\mathbf{R}=0$), single-ion anisotropy (SIA) contributes:

$$ \mathcal{H}_{\text{SIA}} = -\sum_i \mathbf{S}_i \cdot \mathbf{A}_i \cdot \mathbf{S}_i $$

where $\mathbf{A}_i$ is the anisotropy tensor.

5.3 Fourier Transform to Reciprocal Space

5.3.1 Exchange in q-Space

The real-space exchange tensors are transformed to reciprocal space:

$$ \mathbf{J}{ij}(\mathbf{q}) = \sum{\mathbf{R}} \mathbf{J}_{ij}(\mathbf{R}) e^{i\mathbf{q}\cdot\mathbf{R}} $$

This is implemented in `Magnon.Jq()` (see `TB2J/magnon/magnon3.py`):

```
for iR, R in enumerate(Rlist):
    for iqpt, qpt in enumerate(kpoints):
        phase = 2 * np.pi * R @ qpt
        Jq[iqpt] += np.exp(1j * phase) * JRprime[iR]
```

5.3.2 Non-collinear Magnetic Structures

For helimagnetic or spin-spiral systems with propagation vector $\mathbf{Q}$, each exchange tensor is rotated before the Fourier transform:

$$ \mathbf{J}'_{mn}(\mathbf{R}) = \mathbf{R}{m}(\phi)^T \mathbf{J}{mn}(\mathbf{R}) \mathbf{R}_n(\phi) $$

where $\phi = 2\pi \mathbf{R}\cdot\mathbf{Q}$ and $\mathbf{R}(\phi)$ is the rotation matrix.

5.4 Magnon Hamiltonian Construction

5.4.1 Local Reference Frame

For each magnetic atom, we define a local coordinate system where the z'-axis aligns with the spin direction. The rotation from global to local frame is characterized by:

$$ U_i = R_{i,x'} + iR_{i,y'}, \quad V_i = R_{i,z'} $$

where $\mathbf{R}_i$ is the rotation matrix for site i.

5.4.2 Holstein-Primakoff Transformation

In linear spin wave theory, spin operators are expressed in terms of bosonic creation/annihilation operators:

$$ S_i^+ \approx \sqrt{2S_i} a_i, \quad S_i^- \approx \sqrt{2S_i} a_i^\dagger, \quad S_i^z = S_i - a_i^\dagger a_i $$

5.4.3 Dynamical Matrix

The magnon Hamiltonian in the local frame takes the form:

$$ \mathcal{H} = \frac{1}{2}\sum_{\mathbf{q}} \begin{pmatrix} \mathbf{a}\mathbf{q}^\dagger & \mathbf{a}{-\mathbf{q}} \end{pmatrix} \mathbf{H}(\mathbf{q}) \begin{pmatrix} \mathbf{a}\mathbf{q} \ \mathbf{a}{-\mathbf{q}}^\dagger \end{pmatrix} $$

where $\mathbf{a}_\mathbf{q}$ is a vector of bosonic operators for all N magnetic atoms.

The dynamical matrix has the block structure:

$$ \mathbf{H}(\mathbf{q}) = \begin{pmatrix} \mathbf{A}(\mathbf{q}) & \mathbf{B}(\mathbf{q}) \ \mathbf{B}^(\mathbf{-q}) & \mathbf{A}^(\mathbf{-q}) \end{pmatrix} $$

5.4.4 Matrix Elements

The matrix elements are computed from the exchange tensors:

$$ A_{ij}(\mathbf{q}) = \sqrt{S_i S_j} \left[\sum_k V_k^T \mathbf{J}{ik}(\mathbf{0}) V_k S_k \delta{ij} - U_i^* \mathbf{J}_{ij}(\mathbf{q}) U_j \right] $$

$$ B_{ij}(\mathbf{q}) = -\sqrt{S_i S_j} , U_i \mathbf{J}_{ij}(\mathbf{q}) U_j $$

This is implemented in `Magnon.Hq()`:

```
A1 = np.einsum("ix,kijxy,jy->kij", U, Jq, U.conj())
B = np.einsum("ix,kijxy,jy->kij", U, Jq, U)
C = np.diag(np.einsum("ix,ijxy,jy,j->i", V, 2*J0, V, self.Snorm))
H = np.block([[A1 - C, B], [B.swapaxes(-1,-2).conj(), A2 - C]])
```

5.5 Band Structure Calculation

5.5.1 Bogoliubov Transformation

The magnon eigenfrequencies are obtained by diagonalizing the Hamiltonian with the bosonic metric:

$$ \mathbf{g} = \begin{pmatrix} \mathbf{I}_N & 0 \ 0 & -\mathbf{I}_N \end{pmatrix} $$

Using Cholesky decomposition $\mathbf{H} = \mathbf{K}^\dagger \mathbf{K}$, we solve:

$$ \mathbf{K}^\dagger \mathbf{g} \mathbf{K} \mathbf{v} = \omega \mathbf{v} $$

The positive eigenvalues give the magnon energies:

```
K = np.linalg.cholesky(H)
g = np.block([[1*I, 0*I], [0*I, -1*I]])
energies = np.linalg.eigvalsh(K.T.conj() @ g @ K)[:, n:]
```

5.5.2 Goldstone Mode

For ferromagnetic systems, the magnon energy at $\mathbf{q}=0$ should be zero (Goldstone mode). Deviations indicate:

1. Insufficient exchange parameter convergence
2. Incorrect magnetic ground state
3. Missing long-range interactions

5.5.3 k-Path Selection

Band structures are calculated along high-symmetry paths in the Brillouin zone. TB2J supports:

1. **Automatic path detection**: Based on crystal symmetry (default)
2. **Manual specification**: Using string notation (e.g., "GXMR")

```
# Automatic
xlist, kptlist, Xs, knames, spk = auto_kpath(cell, None, npoints=300)

# Manual
bandpath = cell.bandpath(path="GXMR", npoints=300)
```

5.6 Density of States Calculation

5.6.1 Definition

The magnon DOS is defined as:

$$ g(\omega) = \frac{1}{N_k} \sum_{\mathbf{k},n} \delta(\omega - \omega_{n\mathbf{k}}) $$

where $\omega_{n\mathbf{k}}$ is the energy of magnon band n at wavevector $\mathbf{k}$.

5.6.2 Gaussian Smearing

In practice, the delta function is approximated by a Gaussian:

$$ g(\omega) = \frac{1}{N_k} \sum_{\mathbf{k},n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\omega - \omega_{n\mathbf{k}})^2}{2\sigma^2}\right) $$

where σ is the smearing width (typically 0.001 eV 0.08 meV).

5.6.3 k-Mesh Sampling

DOS calculations require dense k-mesh sampling:

```
kpts = monkhorst_pack([20, 20, 20], gamma_center=True)
```

The gamma-centered mesh ensures better convergence for magnetic systems.

5.6.4 Implementation

```
from TB2J.magnon.magnon_dos import MagnonDOSCalculator

calculator = MagnonDOSCalculator(magnon)
```

(continues on next page)

(continued from previous page)

```
calculator.set_kmesh(kmesh=[20, 20, 20], gamma=True)
dos = calculator.get_dos(width=0.001, window=None, npts=401)
```

5.7 Energy Units

TB2J uses consistent energy units:

| Quantity | Internal Unit | Output Unit | |———-|—————|————-| | Exchange parameters | eV | meV | | Magnon energies | eV | meV | | DOS energy axis | eV | meV | | DOS values | states/eV | states/meV |

The conversion factors are applied during plotting:

```
energies_meV = energies_eV * 1000
dos_states_per_meV = dos_states_per_eV / 1000
```

5.8 Validation and Quality Checks

5.8.1 Positive Definiteness

The Hamiltonian should be positive semi-definite at all q-points. If not:

```
try:
    K = np.linalg.cholesky(H)
except np.linalg.LinAlgError:
    min_eig = np.min(np.linalg.eigvalsh(H))
    warn(f"WARNING: The system may be far from the magnetic ground-state. "
         f"Minimum eigenvalue: {min_eig}")
```

5.8.2 Stability Indicators

1. **Zero Goldstone mode**: Expected for ferromagnets
2. **No negative energies**: Indicates ground state stability
3. **Smooth band dispersion**: Validates k-path continuity

5.9 Spin Quantization

In linear spin wave theory, the spin quantum number $S = n/2$ ($n = 1, 2, 3, ...$). TB2J uses magnetic moment values in units of μ_B (Bohr magneton). The input moment corresponds to $2S,\mu_B$, so $S=3/2$ requires $3,\mu_B$.

DFT calculations typically give moments that deviate slightly from these values (e.g., $\sim 3.1,\mu_B$ instead of $3,\mu_B$ for Cr^3). While the DFT value is accurate for exchange calculations, using $2S,\mu_B$ values in magnon calculations ensures internal consistency with the bosonic Holstein-Primakoff transformation.

You can specify custom moments via `--spin-conf` or TOML `spin_conf` (see Magnon Band Structure for usage).

5.10 References

1. Toth, S. & Lake, B. Linear spin wave theory for single-Q incommensurate magnetic structures. *J. Phys.: Condens. Matter* **27**, 166002 (2015).
2. Colpa, J. H. P. Diagonalizing the quadratic boson Hamiltonian with a generalized Bogoliubov transformation. *Physica A* **134**, 417-442 (1986).
3. Mankovsky, S. & Ebert, H. Magnetic exchange interactions and magnon dispersion in multilayer systems from first principles. *Phys. Rev. B* **94**, 144424 (2016).

5.11 See Also

- Magnon Band Structure Usage - Command-line interface and examples
- Exchange Parameters - Input parameters from TB2J calculations
- Tutorial - Step-by-step guide for magnon calculations

CHAPTER

SIX

SPRKKR MAGNON WORKFLOW

TB2J can read the SPRKKR RuO2 reference format and convert it into the existing `SpinIO -> Magnon.load_from_io()` workflow. The first supported input contract is the file set used in `Refs/SPRKKR_RuO2`: `RuO2.str`, `RuO2_JXC_Jij.dat`, and the full-table variant `RuO2_JXC_XCPLTEN_Jij.dat` for parsing/filtering checks.

The MVP does not require `shells.dat`, `Jijatom.*`, potential files, or shape function sidecars. The exchange table columns are interpreted as `IT IQ JT JQ N1 N2 N3 DRX DRY DRZ DR J_xx J_yy J_xy J_yx`; `DRX`, `DRY`, `DRZ`, and `DR` are Cartesian quantities in units of the SPRKKR lattice parameter `A`. Exchange values are read in meV and converted to the eV convention used by the TB2J magnon path.

The reference README states the LKAG convention with prefactor `c = 1`, unit spin direction vectors, and Ru moment `0.5674 mu_B`. Pass the moment explicitly so the magnon normalization is visible in scripts and command lines.

6.1 Python API

```
from TB2J.interfaces import (
    magnon_from_sprkkr,
    read_sprkkr_exchange,
    write_sprkkr_tb2j_results,
)

data = read_sprkkr_exchange(
    structure_file="Refs/SPRKKR_RuO2/RuO2.str",
    exchange_file="Refs/SPRKKR_RuO2/RuO2_JXC_Jij.dat",
    magnetic_species=["Ru"],
    moment=[0.5674, -0.5674],
)

magnon = magnon_from_sprkkr(
    structure_file="Refs/SPRKKR_RuO2/RuO2.str",
    exchange_file="Refs/SPRKKR_RuO2/RuO2_JXC_Jij.dat",
    magnetic_species=["Ru"],
    moment=[0.5674, -0.5674],
    tensor_policy="isotropic",
)

spinio = write_sprkkr_tb2j_results(
    structure_file="Refs/SPRKKR_RuO2/RuO2.str",
    exchange_file="Refs/SPRKKR_RuO2/RuO2_JXC_Jij.dat",
    output_path="TB2J_results",
    magnetic_species=["Ru"],
```

(continues on next page)

(continued from previous page)

```
    moment=[0.5674, -0.5674],
)

labels, bands, xcoords = magnon.get_magnon_bands(path="GX", npoints=20)
```

`write_sprkkr_tb2j_results()` converts SPRKKR inputs into `SpinIO`, writes a full TB2J result directory including `TB2J.pickle`, `exchange.out`, `structure.vasp`, and `Multibinit/exchange.xml`, and returns the in-memory `SpinIO` for immediate use or further inspection. A complete runnable example is available at `examples/magnon/sprkkr_ruo2_tb2j_results.py`.

`moment` can be a single scalar, which is broadcast to all selected magnetic sites, `N` signed scalar values for `N` selected magnetic sites, or `3N` values for explicit moment vectors. Scalar values are placed along the z direction. Values follow the selected SPRKKR magnetic site order. For two Ru sites, use `moment=[0.5674, -0.5674]`; the equivalent explicit-vector form is `moment=[0, 0, 0.5674, 0, 0, -0.5674]`.

Use `tensor_policy="transverse-block"` to place `J_xx`, `J_yy`, `J_xy`, and `J_yx` into the transverse block of a 3x3 tensor with unavailable z-couplings set to zero. Use `tensor_policy="isotropic"` to use `(J_xx + J_yy) / 2` as the scalar exchange. Use `tensor_policy="transverse-block-jzz"` to keep the transverse block and set `J_zz = (J_xx + J_yy) / 2`; all other unavailable entries remain zero. Requests for a full tensor are rejected because the reference files do not contain all tensor components.

6.2 CLI

```
sprkkr2magnon.py \
  -s Refs/SPRKKR_RuO2/RuO2.str \
  -e Refs/SPRKKR_RuO2/RuO2_JXC_Jij.dat \
  -S Ru \
  -m 0.5674 -0.5674 \
  -t isotropic \
  -b \
  -k GX \
  -n 20 \
  --qpoints "G:0,0,0,X:0.5,0,0" \
  -o ruo2_sprkkr_magnon.png
```

The short aliases above are equivalent to `--structure`, `--exchange`, `--magnetic-species`, `--moment`, `--tensor-policy`, `--bands`, `--kpath`, `--npoints`, and `--output`.

Use `--qpoints` to override high-symmetry point coordinates with the same `name:x,y,z` comma-separated format as `TB2J_magnon.py`.

The command writes the plot path and a matching JSON file with the same stem. To write a full TB2J-compatible result directory for later reuse, use:

```
sprkkr2magnon.py \
  -s Refs/SPRKKR_RuO2/RuO2.str \
  -e Refs/SPRKKR_RuO2/RuO2_JXC_Jij.dat \
  -S Ru \
  -m 0.5674 -0.5674 \
  -w TB2J_sprkkr_results
```

CHAPTER

SEVEN

TB2J_EDIT

TB2J_edit is a tool for modifying TB2J results. It allows you to post-process the exchange parameters calculated by TB2J, such as adding single-ion anisotropy, toggling DMI or anisotropic exchange, and symmetrizing the exchange parameters.

7.1 Command Line Interface

The `TB2J_edit.py` command provides several subcommands:

7.1.1 Load and Save

Simply load and save the results (useful for format conversion or checking if the file is valid).

```
TB2J_edit.py load -i TB2J_results/TB2J.pickle -o modified_results
```

7.1.2 Set Single-Ion Anisotropy

Set the single-ion anisotropy (SIA) for specific atomic species.

```
# Set Sm anisotropy to 5 meV along z-axis
TB2J_edit.py set-anisotropy -i TB2J_results/TB2J.pickle -o modified_results -s Sm 5.0 -d
↪"0 0 1" -m
```

- `-s SPECIES K1`: Set the anisotropy constant `K1` for `SPECIES`. Can be used multiple times.
- `-d "x y z"`: Set the direction of the anisotropy axis (default is [0, 0, 1]).
- `-m`: Interpret `K1` in meV (default is eV).

7.1.3 Toggle DMI and Anisotropic Exchange

Enable or disable Dzyaloshinskii-Moriya Interaction (DMI) or anisotropic exchange (Jani).

```
# Disable DMI
TB2J_edit.py toggle-dmi -i TB2J_results/TB2J.pickle -o modified_results --disable

# Enable Anisotropic Exchange
TB2J_edit.py toggle-jani -i TB2J_results/TB2J.pickle -o modified_results --enable

# Disable Isotropic Exchange
TB2J_edit.py toggle-exchange -i TB2J_results/TB2J.pickle -o modified_results --disable
```

7.1.4 Symmetrize Exchange

Symmetrize the isotropic exchange parameters based on a reference crystal structure. This is useful if the calculated J values slightly break symmetry due to numerical noise or if you want to enforce a higher symmetry (e.g., from a high-temperature phase).

```
TB2J_edit.py symmetrize -i TB2J_results/TB2J.pickle -S structure.cif -o modified_results
```

- `-S STRUCTURE`: Path to the reference structure file (e.g., CIF, POSCAR).

7.1.5 Remove Sublattice

Remove all magnetic interactions associated with a specific sublattice. This includes Single-Ion Anisotropy (SIA), Exchange (J), DMI, and Anisotropic Exchange involving atoms of the specified species.

```
TB2J_edit.py remove-sublattice -i TB2J_results/TB2J.pickle -o modified_results -s Sm
```

- `-s SUBLATTICE`: The name of the sublattice (species symbol) to remove (e.g., 'Sm', 'Fe').

7.1.6 Info

Show information about the TB2J results, including composition, number of magnetic atoms, and enabled interactions.

```
TB2J_edit.py info -i TB2J_results/TB2J.pickle
```

7.2 Python API

You can also use the `TB2J.io_exchange.edit` module in your Python scripts.

```python
from TB2J.io_exchange.edit import load, set_anisotropy, set_sia_tensor, remove_sia_
↪tensor, toggle_DMI, toggle_exchange, save

# Load results
spinio = load('TB2J_results/TB2J.pickle')

# Modify results
set_anisotropy(spinio, species='Sm', k1=5.0, k1dir=[0, 0, 1]) # k1 in eV

# Set full 3x3 SIA tensor
import numpy as np
tensor = np.diag([0.001, 0.002, 0.003]) # eV
set_sia_tensor(spinio, species='Sm', tensor=tensor)

toggle_DMI(spinio, enabled=False)
toggle_exchange(spinio, enabled=False)

# Save results
save(spinio, 'modified_results')
```

7.2.1 Functions

- `load(path)`: Load TB2J results from a pickle file.
- `save(spinio, path)`: Save modified results to a directory.

- `set_anisotropy(spinio, species, k1, k1dir)`: Set uniaxial single-ion anisotropy.
- `set_sia_tensor(spinio, species, tensor)`: Set full single-ion anisotropy tensor.
- `remove_sia_tensor(spinio, species)`: Remove single-ion anisotropy tensor.
- `toggle_DMI(spinio, enabled)`: Enable/disable DMI.
- `toggle_Jani(spinio, enabled)`: Enable/disable anisotropic exchange.
- `toggle_exchange(spinio, enabled)`: Enable/disable isotropic exchange.
- `remove_sublattice(spinio, sublattice_name)`: Remove interactions for a sublattice.
- `symmetrize_exchange(spinio, atoms, symprec)`: Symmetrize exchange parameters.

CHAPTER

EIGHT

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.

8.1 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.

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.

8.2 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.

CHAPTER

NINE

ROADMAP OF TB2J

9.1 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

TEN

ECOSYSTEM

10.1 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.

10.2 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.

10.3 Workflows

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

10.4 Codes for Spin Wave methods

- magnopy: magnopy is an open-source python package that analyses spin Hamiltonian (with built-in manipulation of notation) on periodic lattices at several levels of spin wave theory. It is interfaced directly with the TB2J's text output ("exchange.out") and can output magnon band structure, among other thing. It can be used as a python library of within the command line interface.

10.5 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

ELEVEN

FREQUENTLY ASKED QUESTIONS.

11.1 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!

11.2 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.

11.3 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.

11.4 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*>

11.5 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.

11.6 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.

11.7 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.

11.8 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.

11.9 Does TB2J work with 2D structures or molecules?

Yes.

CHAPTER TWELVE

CONTRIBUTORS

The TB2J package is initially developed:

- 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 Rybakov (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

THIRTEEN

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

FOURTEEN

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.

14.1 Code

14.1.1 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

FIFTEEN

RELEASE NOTES

15.1 Current development version (v1.0.0-alpha)

These are the new features and changes not yet included in the official release.

- Computing MAE and single-ion anisotropy is now possible with the ABACUS and SIESTA interfaces. This currently requires an non-official SIESTA branch which can seperate the spin-orbit coupling and the exchange-correlation Hamiltonian. (see this MR: https://gitlab.com/siesta-project/siesta/-/merge_requests/309)
- The full implementation of the magnon band structure from the linear spin wave theory. (Thanks to Andres Tellez Mora and Aldo Romero!)
- An improved method of the downfolding method which implements the ligand correction to the exchange based on the Wannier function method. This requires the updated version of LaWaF (https://github.com/mailhexu/lawaf) and the updated version of the TB2J_downfold.py script.
- Added a new parameter (–index_magnetic_atoms) to the wann2J.py, siesta2J.py, and abacus2J.py scripts, so that a more detailed selection of the magnetic atoms than the one in –element is possible. This is useful when we need to select a subset of a specific element in a system.
- There is a major refactoring of the interface to the DFT codes. The parsing of the electron Hamiltonian from the DFT codes are now in a separate python package called HamiltonIO (github.com/mailhexu/HamiltonIO). This package is used by TB2J but is made general to be used with other packages too.

15.2 v0.11.0 October 10, 2024

- Allowing to symmetrize the exchange according to the crystal symmetry with the TB2J_symmetrize.py script. Note that the spin order is not considered in this symmetrization process.

15.3 v0.10.0 September 1, 2024

- Improved orbital-decomposition to the ABACUS interface.
- Allow computing anisotropic J and DMI without three or more calculations within ABACUS and SIESTA interfaces.

15.4 v0.9.0 March 22, 2024

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

15.5 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)

15.6 v0.8.1 Febrary 25, 2024

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

15.7 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.

15.8 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

15.9 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.

15.10 v0.7.3.1 July 24, 2022

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

15.11 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).

15.12 v0.7.2.1 April 20, 2022

Fix compatibility issue with Python3.10

15.13 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.

15.14 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.)

15.15 v0.7.0 October 05, 2021

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

15.16 v0.6.10 September 29, 2021

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

15.17 v0.6.9 September 15, 2021

Better xticks in magnon bands.

15.18 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.

15.19 v0.6.7 September 10, 2021

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

15.20 v0.6.6 September 1, 2021

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

15.21 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.

15.22 v0.6.3 July 3, 2021

Revert qsolver to old version.

15.23 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.

15.24 v0.6.1

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

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

15.25 v0.6.0

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

15.26 v0.5.0

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

15.27 v0.4.4 March 16, 2021

Allow parallel over k in tight binding eigen solver.

15.28 v0.4.3 March 11, 2021

Add Reference to TB2J paper.

15.29 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.

15.30 v0.4.1 February 2, 2021

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

15.31 v0.4.0 February 1, 2021

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

15.32 v0.3.8 December 29, 2020

Add --output_path option to specify the output path.

15.33 v0.3.6 December 7, 2020

Use Simpson's rule instead of Euler for integration.

15.34 v0.3.5 November 3, 2020

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

15.35 v0.3.3 September 12, 2020

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

15.36 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.
```

15.37 v0.3.1 September 3, 2020

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

15.38 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

15.39 v0.2 2020

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

15.40 v0.1 2018

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

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex