ABACUS

ABACUS

Apr 24, 2024

QUICK START

ABACUS (Atomic-orbital Based Ab-initio Computation at UStc) is an open-source computer code package based on density functional theory (DFT). The package utilizes both plane wave and numerical atomic basis sets with the usage of norm-conserving pseudopotentials to describe the interactions between nuclear ions and valence electrons. ABACUS supports LDA, GGA, meta-GGA, and hybrid functionals. Apart from single-point calculations, the package allows geometry optimizations and ab-initio molecular dynamics with various ensembles. The package also provides a variety of advanced functionalities for simulating materials, including the DFT+U, VdW corrections, and implicit solvation model, etc. In addition, ABACUS strives to provide a general infrastructure to facilitate the developments and applications of novel machine-learning-assisted DFT methods (DeePKS, DP-GEN, DeepH, etc.) in molecular and material simulations.

CHAPTER

ONE

EASY INSTALLATION

This guide helps you install ABACUS with basic features. **For DeePKS, DeePMD and Libxc support, or building with `make`, please refer to *the advanced installation guide*** after going through this page. We recommend building ABACUS with `cmake` to avoid dependency issues. We recommend compiling ABACUS(and possibly its requirements) from the source code using the latest compiler for the best performace. You can also deploy ABACUS **without building** by *Docker* or *conda*. Please note that ABACUS only supports Linux; for Windows users, please consider using WSL or docker.

1.1 Prerequisites

To compile ABACUS, please make sure that the following prerequisites are present:

- CMake >= 3.16 .
- C++ compiler, supporting C++11. You can use Intel® C++ compiler or GCC.

  GCC version 5 or later is always required. Intel compilers also use GCC headers and libraries(ref).
- MPI library. The recommended versions are Intel MPI, MPICH or Open MPI.
- Fortran compiler if you are building `BLAS`, `LAPACK`, `ScaLAPACK`, and `ELPA` from source file. You can use Intel® Fortran Compiler or GFortran.
- BLAS. You can use OpenBLAS.
- LAPACK.
- FFTW3.

These requirements support the calculation of plane-wave basis in ABACUS. For LCAO basis calculation, additional components are required:

- ScaLAPACK.
- CEREAL.
- ELPA >= 2017 (optional).

1.2 Install requirements

Some of these packages can be installed with popular package management system, such as `apt` and `yum`:

```
sudo apt update && sudo apt install -y libopenblas-openmp-dev liblapack-dev␣
↪libscalapack-mpi-dev libelpa-dev libfftw3-dev libcereal-dev libxc-dev g++ make␣
↪cmake bc git pkgconf
```

> Installing ELPA by apt only matches requirements on Ubuntu 22.04. For earlier linux distributions, you should build ELPA from source.

We recommend Intel® oneAPI toolkit (former Intel® Parallel Studio) as toolchain. The Intel® oneAPI Base Toolkit contains Intel® oneAPI Math Kernel Library (aka `MKL`), including `BLAS`, `LAPACK`, `ScaLAPACK` and `FFTW3`. The Intel® oneAPI HPC Toolkit contains Intel® MPI Library, and C++ compiler(including MPI compiler).

> Please note that building `elpa` with a different MPI library may cause conflict. Don't forget to set environment variables before you start! `cmake` will use Intel MKL if the environment variable `MKLROOT` is set.

Please refer to our guide on installing requirements.

1.3 Install requirements by toolchain

We offer a set of toolchain scripts to compile and install all the requirements automatically and suitable for machine characteristic in an online or offline way. The toolchain can be downloaded with ABACUS repo, which is easily used and can have a convenient installation under HPC environment in both `GNU` or `Intel-oneAPI` toolchain. Sometimes, ABACUS by toolchain installation may have highly efficient performance. A Tutorial for using this toolchain can be accessed in bohrium-notebook

> Notice: the toolchain is under development, please let me know if you encounter any problem in using this toolchain.

1.4 Get ABACUS source code

Of course a copy of ABACUS source code is required, which can be obtained via one of the following choices:

- Clone the whole repo with git: `git clone https://github.com/deepmodeling/abacus-develop.git`
- Clone the minimum required part of repo: `git clone https://github.com/deepmodeling/abacus-develop.git --depth=1`
- Download the latest source code without git: `wget https://github.com/deepmodeling/abacus-develop/archive/refs/heads/develop.zip`
- Get the source code of a stable version here
- If you have connection issues accessing GitHub, please try out our official Gitee repo: e.g. `git clone https://gitee.com/deepmodeling/abacus-develop.git`

1.4.1 Update to latest release

Please check the release page for the release note of a new version.

It is OK to download the new source code from beginning following the previous step.

To update your cloned git repo in-place:

```
git remote -v
# Check if the output contains the line below
# origin https://github.com/deepmodeling/abacus-develop.git (fetch)
# The remote name is marked as "upstream" if you clone the repo from your own fork.

# Replace "origin" with "upstream" or the remote name corresponding to deepmodeling/
↪abacus-develop if necessary
git fetch origin
git checkout v3.2.0 # Replace the tag with the latest version
git describe --tags # Verify if the tag has been successfully checked out
```

Then proceed to the *Build and Install* part. If you encountered errors, try remove the `build` directory first and reconfigure.

To use the codes under active development:

```
git checkout develop
git pull
```

1.5 Configure

The basic command synopsis is:

```
cd abacus-develop
cmake -B build [-D <var>=<value>] ...
```

Here, 'build' is the path for building ABACUS; and '-D' is used for setting up some variables for CMake indicating optional components or requirement positions.

- `CMAKE_INSTALL_PREFIX`: the path of ABACUS binary to install; `/usr/local/bin/abacus` by default
- Compilers
  - `CMAKE_CXX_COMPILER`: C++ compiler; usually `g++`(GNU C++ compiler) or `icpx`(Intel C++ compiler). Can also set from environment variable `CXX`. It is OK to use MPI compiler here.
  - `MPI_CXX_COMPILER`: MPI wrapper for C++ compiler; usually `mpicxx` or `mpiicpc`(for Intel MPI).
- Requirements: Unless indicated, CMake will try to find under default paths.
  - `MKLROOT`: If environment variable `MKLROOT` exists, `cmake` will take MKL as a preference, i.e. not using `LAPACK`, `ScaLAPACK` and `FFTW`. To disable MKL, unset environment variable `MKLROOT`, or pass `-DMKLROOT=OFF` to `cmake`.
  - `LAPACK_DIR`: Path to OpenBLAS library `libopenblas.so`(including BLAS and LAPACK)
  - `SCALAPACK_DIR`: Path to ScaLAPACK library `libscalapack.so`
  - `ELPA_DIR`: Path to ELPA install directory; should be the folder containing 'include' and 'lib'.

Note: In ABACUS v3.5.1 or earlier, if you install ELPA from source , please add a symlink to avoid the additional include file folder with version name: `ln -s elpa/include/elpa-2021.05.002/elpa elpa/include/elpa` to help the build system find ELPA headers.

- `FFTW3_DIR`: Path to FFTW3.
- `CEREAL_INCLUDE_DIR`: Path to the parent folder of `cereal/cereal.hpp`. Will download from GitHub if absent.
- `Libxc_DIR`: (Optional) Path to Libxc.

  Note: In ABACUS v3.5.1 or earlier, Libxc built from source with Makefile is NOT supported; please compile Libxc with CMake instead.
- `LIBRI_DIR`: (Optional) Path to LibRI.
- `LIBCOMM_DIR`: (Optional) Path to LibComm.

- Components: The values of these variables should be 'ON', '1' or 'OFF', '0'. The default values are given below.
  - `ENABLE_LCAO=ON`: Enable LCAO calculation. If SCALAPACK, ELPA or CEREAL is absent and only require plane-wave calculations, the feature of calculating LCAO basis can be turned off.
  - `ENABLE_LIBXC=OFF`: *Enable Libxc* to suppport variety of functionals. If `Libxc_DIR` is defined, `ENABLE_LIBXC` will set to 'ON'.
  - `ENABLE_LIBRI=OFF`: *Enable LibRI* to suppport variety of functionals. If `LIBRI_DIR` and `LIBCOMM_DIR` is defined, `ENABLE_LIBRI` will set to 'ON'.
  - `USE_OPENMP=ON`: Enable OpenMP support. Building ABACUS without OpenMP is not fully tested yet.
  - `BUILD_TESTING=OFF`: *Build unit tests*.
  - `ENABLE_GOOGLEBENCH=OFF`: *Build performance tests*
  - `ENABLE_MPI=ON`: Enable MPI parallel compilation. If set to `OFF`, a serial version of ABACUS with PW basis only will be compiled. Currently serial version of ABACUS with LCAO basis is not supported yet, so `ENABLE_LCAO` will be automatically set to `OFF`.
  - `ENABLE_COVERAGE=OFF`: Build ABACUS executable supporting *coverage analysis*. This feature has a drastic impact on performance.
  - `ENABLE_ASAN=OFF`: Build with Address Sanitizer. This feature would help detecting memory problems.
  - `USE_ELPA=ON`: Use ELPA library in LCAO calculations. If this value is set to OFF, ABACUS can be compiled without ELPA library.

Here is an example:

```
CXX=mpiicpc cmake -B build -DCMAKE_INSTALL_PREFIX=~/abacus -DELPA_DIR=~/elpa-2016.05.
↪004/build -DCEREAL_INCLUDE_DIR=~/cereal/include
```

1.6 Build and Install

After configuring, build and install by:

```
cmake --build build -j`nproc`
cmake --install build
```

You can change the number after `-j` on your need: set to the number of CPU cores(`nproc`) to reduce compilation time.

1.7 Run

If ABACUS is installed into a custom directory using `CMAKE_INSTALL_PREFIX`, please add it to your environment variable `PATH` to locate the correct executable. *(Note: `my-install-dir` should be changed to the location of your installed abacus:`/home/your-path/abacus/bin/`.)*

```
export PATH=/my-install-dir/:$PATH
```

Please set OpenMP threads by setting environment variable:

```
export OMP_NUM_THREADS=1
```

Enter a directory containing a `INPUT` file. Please make sure structure, pseudo potential, or orbital files indicated by `INPUT` is at the correct location.

```
cd abacus-develop/examples/force/pw_Si2
```

Use 4 MPI processes to run, for example:

```
mpirun -n 4 abacus
```

> The total thread count(i.e. OpenMP per-process thread count * MPI process count) should not exceed the number of cores in your machine.

Please refer to *hands-on guide* for more instructions.

> Note: Some Intel CPU has a feature named Hyper-Threading(HT). This feature enables one physical core switch fastly between two logical threads. It would benefits from I/O bound tasks: when a thread is blocked by I/O, the CPU core can work on another thread. However, it helps little on CPU bound tasks, like ABACUS and many other scientific computing softwares. We recommend using the physical CPU core number. To determine if HT is turned on, execute `lscpu | grep 'per core'` and see if 'Thread(s) per core' is 2.

1.8 Container Deployment

> Please note that containers target at developing and testing, but not massively parallel computing for production. Docker has a bad support to MPI, which may cause performance degradation.

We've built a ready-for-use version of ABACUS with docker here. For a quick start: pull the image, prepare the data, run container. Instructions on using the image can be accessed in Dockerfile. A mirror is available by `docker pull registry.dp.tech/deepmodeling/abacus`.

We also offer a pre-built docker image containing all the requirements for development. Please refer to our Package Page.

The project is ready for VS Code development container. Please refer to Developing inside a Container. Choose `Open a Remote Window -> Clone a Repository in Container Volume` in VS Code command palette, and put the git address of `ABACUS` when prompted.

For online development environment, we support GitHub Codespaces: Create a new Codespace

We also support Gitpod: Open in Gitpod

1.9 Install by conda

Conda is a package management system with a separated environment, not requiring system privileges. You can refer to DeepModeling conda FAQ for how to setup a conda environment. A pre-built ABACUS binary with all requirements is available at conda-forge. It supports advanced features including Libxc, LibRI, and DeePKS. Conda will install the GPU-supported version of ABACUS if a valid GPU driver is present. Please refer to *the advanced installation guide* for more details.

```
# Install
# We recommend installing ABACUS in a new environment to avoid potential conflicts:
conda create -n abacus_env abacus "libblas=*=*mkl" mpich -c conda-forge

# Run
conda activate abacus_env
OMP_NUM_THREADS=1 mpirun -n 4 abacus

# Update
conda update -n abacus_env abacus -c conda-forge
```

> If OpenBLAS gives warning about OpenMP threads, please install conda package `"openblas=*=openmp*"` or `"libblas=*=*mkl"`. See switching BLAS implementation in conda.
>
> ABACUS supports `OpenMPI` and `MPICH` variant. Install `mpich` or `openmpi` package to switch MPI library if required.

For more details on building a conda package of ABACUS locally, please refer to the conda recipe file.

> Note: The deepmodeling conda channel offers historical versions of ABACUS.

1.9.1 Developing with conda

It is possible to build ABACUS from source based on the conda environment.

```
conda create -n abacus_env abacus -c conda-forge
conda activate abacus_env
export CMAKE_PREFIX_PATH=$CONDA_PREFIX:$CMAKE_PREFIX_PATH

# By default OpenBLAS is used; run `conda install "blas=*=mkl" mkl_fft mkl-devel -c␣
↪conda-forge` to switch implementation.
export MKLROOT=$CONDA_PREFIX # If Intel MKL is required.

export CMAKE_PREFIX_PATH=`python -c 'import torch;print(torch.utils.cmake_prefix_path)
↪'`:$CMAKE_PREFIX_PATH # If DEEPKS support is required;
# usually expands to `$CONDA_PREFIX/lib/python3.1/site-packages/torch/share/cmake`
```

And, follow the instructions in *Build and Install* part above withou manually setting paths to dependencies. See *the advanced installation guide* for more features. Make sure the environment variables are set before running `cmake`. Possible command: `cmake -B build -DENABLE_DEEPKS=ON -DENABLE_LIBXC=ON -DENABLE_LIBRI=ON`.

CHAPTER

TWO

TWO QUICK EXAMPLES

2.1 Running SCF Calculation

2.1.1 A quick LCAO example

ABACUS is well known for its support of LCAO (Linear Combination of Atomic Orbital) basis set in calculating periodic condensed matter systems, so it's a good choice to start from a LCAO example of self-consistent field (SCF) calculation. Here, FCC MgO has been chosen as a quick start example. The default name of a structure file in ABACUS is `STRU`. The `STRU` file for FCC MgO in a LCAO calculation is shown below:

```
#This is the atom file containing all the information
#about the lattice structure.

ATOMIC_SPECIES
Mg 24.305  Mg_ONCV_PBE-1.0.upf  # element name, atomic mass, pseudopotential file
O  15.999 O_ONCV_PBE-1.0.upf

NUMERICAL_ORBITAL
Mg_gga_8au_100Ry_4s2p1d.orb
O_gga_8au_100Ry_2s2p1d.orb

LATTICE_CONSTANT
1.8897259886                # 1.8897259886 Bohr =  1.0 Angstrom

LATTICE_VECTORS
4.25648 0.00000 0.00000
0.00000 4.25648 0.00000
0.00000 0.00000 4.25648

ATOMIC_POSITIONS
Direct                  #Cartesian(Unit is LATTICE_CONSTANT)
Mg                      #Name of element
0.0                     #Magnetic for this element.
4                       #Number of atoms
0.0  0.0  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.5  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.0  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.5  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
O                       #Name of element
0.0                     #Magnetic for this element.
4                       #Number of atoms
0.5  0.0  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.5  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
```

(continues on next page)

(continued from previous page)

```
0.0  0.0  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.5  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
```

Next, the `INPUT` file is required, which sets all key parameters to direct ABACUS how to calculte and what to output:

```
INPUT_PARAMETERS
suffix                  MgO
ntype                   2
pseudo_dir              ./
orbital_dir                ./
ecutwfc                 100             # Rydberg
scf_thr                 1e-4                # Rydberg
basis_type              lcao
calculation             scf                 # this is the key parameter telling abacus␣
↪to do a scf calculation
```

The pseudopotential files of `Mg_ONCV_PBE-1.0.upf` and `O_ONCV_PBE-1.0.upf` should be provided under the directory of `pseudo_dir` defined in `INPUT` (the default directory is "./"), and the orbital files `Mg_gga_8au_100Ry_4s2p1d.orb` and `O_gga_8au_100Ry_2s2p1d.orb` under the directory of `orbital_dir` also defined in `INPUT` (the default directory is "./"). The pseudopotential and orbital files can be downloaded from the ABACUS website.

The final mandatory input file is called `KPT`, which sets the reciprocal space k-mesh. Below is an example:

```
K_POINTS
0
Gamma
4 4 4 0 0 0
```

After all the above input files have been set, one should be able to run the first quick example. The simplest way is to use the command line, e.g.:

```
mpirun -np 2 abacus
```

The main output information is stored in the file `OUT.MgO/running_scf.log`, which starts with

```
                            WELCOME TO ABACUS v3.2

              'Atomic-orbital Based Ab-initio Computation at UStc'

                     Website: http://abacus.ustc.edu.cn/

   Version: Parallel, in development
   Processor Number is 2
   Start Time is Mon Oct 24 01:47:54 2022

 ------------------------------------------------------------------------------------

 READING GENERAL INFORMATION
                           global_out_dir = OUT.MgO/
                           global_in_card = INPUT
                               pseudo_dir =
                              orbital_dir =
                                    DRANK = 1
                                    DSIZE = 2
                                   DCOLOR = 1
```

(continues on next page)

(continued from previous page)

```
                                    GRANK = 1
                                    GSIZE = 1
 The esolver type has been set to : ksdft_lcao




 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 |                                                                            |
 | Reading atom information in unitcell:                                      |
 | From the input file and the structure file we know the number of           |
 | different elments in this unitcell, then we list the detail                |
 | information for each element, especially the zeta and polar atomic         |
 | orbital number for each element. The total atom number is counted.         |
 | We calculate the nearest atom distance for each atom and show the          |
 | Cartesian and Direct coordinates for each atom. We list the file           |
 | address for atomic orbitals. The volume and the lattice vectors            |
 | in real and reciprocal space is also shown.                                |
 |                                                                            |
 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

......
```

If ABAUCS finishes successfully, the total energy will be output in `OUT.MgO/running_scf.log`:

```
 ---------------------------------------------
 !FINAL_ETOT_IS -7663.897267807250 eV
 ---------------------------------------------
```

2.1.2 A quick PW example

In order to run a SCF calculation with PW (Plane Wave) basis set, one has only to change the tag `basis_type` from `lcao` to `pw` in the `INPUT` file, and no longer needs to provide orbital files under `NUMERICAL_ORBITAL` in the `STRU` file.

The `INPUT` file follows as:

```
INPUT_PARAMETERS
suffix                  MgO
ntype                   2
pseudo_dir              ./
ecutwfc                 100             # Rydberg
scf_thr                 1e-4              # Rydberg
basis_type              pw              # changes the type of basis set
calculation             scf               # this is the key parameter telling abacus␣
↪to do a scf calculation
```

And the `STRU` file will be:

```
#This is the atom file containing all the information
#about the lattice structure.

ATOMIC_SPECIES
Mg 24.305  Mg_ONCV_PBE-1.0.upf  # element name, atomic mass, pseudopotential file
```

(continues on next page)

(continued from previous page)

```
O  15.999 O_ONCV_PBE-1.0.upf

LATTICE_CONSTANT
1.8897259886                # 1.8897259886 Bohr =  1.0 Angstrom

LATTICE_VECTORS
4.25648 0.00000 0.00000
0.00000 4.25648 0.00000
0.00000 0.00000 4.25648

ATOMIC_POSITIONS
Direct                  #Cartesian(Unit is LATTICE_CONSTANT)
Mg                      #Name of element
0.0                     #Magnetic for this element.
4                       #Number of atoms
0.0  0.0  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.5  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.0  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.5  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
O                       #Name of element
0.0                     #Magnetic for this element.
4                       #Number of atoms
0.5  0.0  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
0.5  0.5  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.0  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.5  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
```

Use the same pseudopotential and `KPT` files as the above LCAO example. The final total energy will be output:

```
 ----------------------------------------
 !FINAL_ETOT_IS -7665.688319476949 eV
 ----------------------------------------
```

2.2 Running Geometry Optimization

In order to run a full geometry optimization in ABACUS, the tag `calculation` in `INPUT` should be set to `cell-relax`. In addition, the convergence criteria for atomics force and cell stress can be set through the tags `force_thr_ev` and `stress_thr`, respectively. The maximum number of ionc steps is controlled by `relax_nmax`.

2.2.1 A quick LCAO example

The `INPUT` is provided as follows:

```
INPUT_PARAMETERS
suffix                  MgO
ntype                   2
nelec                   0.0
pseudo_dir              ./
orbital_dir             ./
ecutwfc                 100             # Rydberg
scf_thr                 1e-4                # Rydberg
basis_type              lcao
```

(continues on next page)

(continued from previous page)

```
calculation             cell-relax        # this is the key parameter telling abacus␣
↪to do a optimization calculation
force_thr_ev                0.01                # the threshold of the force␣
↪convergence, in unit of eV/Angstrom
stress_thr                  5                 # the threshold of the stress convergence,␣
↪in unit of kBar
relax_nmax                  100                 # the maximal number of ionic iteration␣
↪steps
out_stru                1
```

Use the same `KPT`, `STRU`, pseudopotential, and orbital files as in the above SCF-LCAO example. The final optimized structure can be found in `STRU_NOW.cif` and `OUT.MgO/running_cell-relax.log`.

2.2.2 A quick PW example

The `INPUT` is provided as follows:

```
INPUT_PARAMETERS
suffix                  MgO
ntype                   2
nelec                   0.0
pseudo_dir              ./
ecutwfc                 100             # Rydberg
scf_thr                 1e-4              # Rydberg
basis_type              pw
calculation             cell-relax        # this is the key parameter telling abacus␣
↪to do a optimization calculation
force_thr_ev                0.01                # the threshold of the force␣
↪convergence, in unit of eV/Angstrom
stress_thr                  5                 # the threshold of the stress convergence,␣
↪in unit of kBar
relax_nmax                  100                 # the maximal number of ionic iteration␣
↪steps
out_stru                1
```

Use the same `KPT`, `STRU`, and pseudopotential files as in the above SCF-PW examples. The final optimized structure can be found in `STRU_NOW.cif` and `OUT.MgO/running_cell-relax.log`.

CHAPTER

THREE

BRIEF INTRODUCTION OF THE INPUT FILES

The following files are the central input files for ABACUS. Before executing the program, please make sure these files are prepared and stored in the working directory. Here we give some simple descriptions. For more details, users should consult the Advanced session.

3.1 *INPUT*

The `INPUT` file contains parameters that control the type of calculation as well as a variety of settings.

Below is an example `INPUT` file with some of the most important parameters that need to be set:

```
INPUT_PARAMETERS
suffix                  MgO
ntype                   2
pseudo_dir              ./
orbital_dir                ./
ecutwfc                 100             # Rydberg
scf_thr                 1e-4               # Rydberg
basis_type              lcao
calculation             scf                # this is the key parameter telling abacus␣
↪to do a scf calculation
out_chg                         True
```

The parameter list always starts with key word `INPUT_PARAMETERS`. Any content before `INPUT_PARAMETERS` will be ignored.

Any line starting with `#` or `/` will also be ignored.

Each parameter value is provided by specifying the name of the input variable and then putting the value after the name, separated by one or more blank characters(space or tab). The following characters (> 150) in the same line will be neglected.

Depending on the input variable, the value may be an integer, a real number or a string. The parameters can be given in any order, but only one parameter should be given per line.

Furthermore, if a given parameter name appeared more than once in the input file, only the last value will be taken.

> **Note:** if a parameter name is not recognized by the program, the program will stop with an error message.

In the above example, the meanings of the parameters are:

- `suffix` : the name of the system, default `ABACUS`
- `ntype` : how many types of elements in the unit cell
- `pseudo_dir` : the directory where pseudopotential files are provided

- `orbital_dir` : the directory where orbital files are provided
- `ecutwfc` : the plane-wave energy cutoff for the wave function expansion (UNIT: Rydberg)
- `scf_thr` : the threshold for the convergence of charge density (UNIT: Rydberg)
- `basis_type` : the type of basis set for expanding the electronic wave functions
- `calculation` : the type of calculation to be performed by ABACUS
- `out_chg` : if true, output thee charge density oon real space grid

For a complete list of input parameters, please consult this *instruction*.

> **Note:** Users cannot change the filename "INPUT" to other names. Boolean paramerters such as `out_chg` can be set by using `True` and `False`, `1` and `0`, or `T` and `F`. It is case insensitive so that other preferences such as `true` and `false`, `TRUE` and `FALSE`, and `t` and `f` for setting boolean values are also supported.

3.2 *STRU*

The structure file contains structural information about the system, e.g., lattice constant, lattice vectors, and positions of the atoms within a unit cell. The positions can be given either in direct or Cartesian coordinates.

An example of the `STRU` file is given as follows :

```
#This is the atom file containing all the information
#about the lattice structure.

ATOMIC_SPECIES
Mg 24.305  Mg_ONCV_PBE-1.0.upf  # element name, atomic mass, pseudopotential file
O  15.999 O_ONCV_PBE-1.0.upf

NUMERICAL_ORBITAL
Mg_gga_8au_100Ry_4s2p1d.orb
O_gga_8au_100Ry_2s2p1d.orb

LATTICE_CONSTANT
1.8897259886                # 1.8897259886 Bohr =  1.0 Angstrom

LATTICE_VECTORS
4.25648 0.00000 0.00000
0.00000 4.25648 0.00000
0.00000 0.00000 4.25648

ATOMIC_POSITIONS
Direct                    #Cartesian(Unit is LATTICE_CONSTANT)
Mg                        #Name of element
0.0                       #Magnetic for this element.
4                         #Number of atoms
0.0  0.0  0.0  0 0 0      #x,y,z, move_x, move_y, move_z
0.0  0.5  0.5  0 0 0      #x,y,z, move_x, move_y, move_z
0.5  0.0  0.5  0 0 0      #x,y,z, move_x, move_y, move_z
0.5  0.5  0.0  0 0 0      #x,y,z, move_x, move_y, move_z
O                         #Name of element
0.0                       #Magnetic for this element.
4                         #Number of atoms
0.5  0.0  0.0  0 0 0      #x,y,z, move_x, move_y, move_z
0.5  0.5  0.5  0 0 0      #x,y,z, move_x, move_y, move_z
```

(continues on next page)

(continued from previous page)

```
0.0  0.0  0.5  0 0 0    #x,y,z, move_x, move_y, move_z
0.0  0.5  0.0  0 0 0    #x,y,z, move_x, move_y, move_z
```

Note: users may choose a different name for their structure file using the keyword `stru_file`. The order of the pseudopotential file list and the numerical orbital list (if LCAO is applied) MUST be consistent with that of the atomic type given in `ATOMIC_POSITIONS`.

For a more detailed description of STRU file, please consult *here*.

3.3 *KPT*

This file contains information of the kpoint grid setting for the Brillouin zone sampling.

An example of the `KPT` file is given below:

```
K_POINTS
0
Gamma
4 4 4 0 0 0
```

Note: users may choose a different name for their k-point file using keyword `kpoint_file`

For a more detailed description, please consult *here*.

- The pseudopotential files

  Norm-conserving pseudopotentials are used in ABACUS, in the UPF file format.The filename of each element's pseudopotential needs to be specified in the STRU file, together with the directory of the pseudopotential files unless they are already present in the working directory.

  More information on pseudopotentials is given *here*.

- The numerical orbital files

  This part is only required in LCAO calculations. The filename for each element's numerical orbital basis needs to be specified in the STRU file, together with the directory of the orbital files unless they are already present in the working directory. ABACUS provides numerical atomic basis sets of different accuracy levels for most elements commonly used. Users can download these basis sets from the website. Moreover, users can generate numerical atomic orbitals by themselves, and the procedure is provided in this *short introduction*.

CHAPTER

FOUR

BRIEF INTRODUCTION OF THE OUTPUT FILES

The following files are the central output files for ABACUS. After executing the program, you can obtain a log file containing the screen output, more detailed outputs are stored in the working directory `OUT.suffix` (Default one is `OUT.ABACUS`). Here we give some simple descriptions.

4.1 *INPUT*

Different from `INPUT` given by the users, `OUT.suffix/INPUT` contains all parameters in ABACUS.

> **Note:** `OUT.suffix/INPUT` contain the initial default of ABACUS instead of the real parameters used in calculations. If you want to figure out the real parameters used in calculations, you can open `OUT.suffix/runing_scf.log` and research corresponding parameter you are interested.

For a complete list of input parameters, please consult this *instruction*.

4.2 *running_scf.log*

`running_scf.log` contains information on nearly all function calls made during the execution of ABACUS.

4.3 *KPT*

This file contains the information of all generated k-points, as well as the list of k-points actually used for calculations after considering symmetry.

4.4 *istate.info*

This file includes the energy levels computed for all k-points. From left to right, the columns represent: energy level index, eigenenergy, and occupancy number.

Below is an example `istate.info`:

```
BAND              Energy(ev)              Occupation              Kpoint = 1      ␣
↪                  (0 0 0)
      1               -5.33892                 0.03125
      2                6.68535               0.0312006
      3                6.68535               0.0312006
```

(continues on next page)

(continued from previous page)

```
      4                6.68535              0.0312006
      5                9.41058                      0
```

4.5 *STRU_SIMPLE.cif*

ABACUS generates a `.cif` format structure file based on the input file `STRU`, facilitating users to visualize with commonly used software. `STRU_READIN_ADJUST.cif` is the structure after considering symmetry.

4.6 *warning.log*

The file contains all the warning messages generated during the ABACUS run.

CHAPTER

FIVE

ADVANCED INSTALLATION OPTIONS

This guide helps you install ABACUS with advanced features. Please make sure to read the *easy-installation guide* before.

5.1 Build with Libxc

ABACUS use exchange-correlation functionals by default. However, for some functionals (such as HSE hybrid functional), Libxc is required.

Dependency: Libxc >= 5.1.7 .

> Note: Building Libxc from source with Makefile does NOT support using it in CMake here. Please compile Libxc with CMake instead.

If Libxc is not installed in standard path (i.e. installed with a custom prefix path), you can set `Libxc_DIR` to the corresponding directory.

```
cmake -B build -DLibxc_DIR=~/libxc
```

5.2 Build with DeePKS

If DeePKS feature is required for DeePKS-kit, the following prerequisites and steps are needed:

- C++ compiler, supporting **C++14** (GCC >= 5 is sufficient)
- CMake >= 3.18
- LibTorch with cxx11 ABI supporting CPU
- Libnpy

```
cmake -B build -DENABLE_DEEPKS=1 -DTorch_DIR=~/libtorch/share/cmake/Torch/ -Dlibnpy_
↪INCLUDE_DIR=~/libnpy/include
```

> CMake will try to download Libnpy if it cannot be found locally.

5.3 Build with DeePMD-kit

> Note: This part is only required if you want to load a trained DeeP Potential and run molecular dynamics with that. To train the DeeP Potential with DP-GEN, no extra prerequisite is needed and please refer to this page for ABACUS interface with DP-GEN.

If the Deep Potential model is employed in Molecule Dynamics calculations, the following prerequisites and steps are needed:

- DeePMD-kit
- TensorFlow

```
cmake -B build -DDeePMD_DIR=~/deepmd-kit -DTensorFlow_DIR=~/tensorflow
```

> `deepmd_c`/`deepmd_cc` and `tensorflow_cc` libraries would be called according to `DeePMD_DIR` and `TensorFlow_DIR`, which is showed in detail in this page.

5.4 Build with LibRI and LibComm

The new EXX implementation depends on two external libraries:

- LibRI
- LibComm

These two libraries are added as submodules in the deps folder. Set `-DENABLE_LIBRI=ON` to build with these two libraries.

If you prefer using manually downloaded libraries, provide `-DLIBRI_DIR=${path to your LibRI folder} -DLIBCOMM_DIR=${path to your LibComm folder}`.

5.5 Build Unit Tests

To build tests for ABACUS, define `BUILD_TESTING` flag. You can also specify path to local installation of Googletest by setting `GTEST_DIR` flags. If not found in local, the configuration process will try to download it automatically.

```
cmake -B build -DBUILD_TESTING=1
```

After building and installing, unit tests can be performed with `ctest`.

To run a subset of unit test, use `ctest -R <test-match-pattern>` to perform tests with name matched by given pattern.

5.6 Build Performance Tests

To build performance tests for ABACUS, define `ENABLE_GOOGLEBENCH` flag. You can also specify the path to a local installation of Google Benchmark by setting `BENCHMARK_DIR` flags. If not found locally, the configuration process will try to download it automatically.

```
cmake -B build -DENABLE_GOOGLEBENCH=1
```

Google Benchmark requires Google Test to build and run the tests. When setting `ENABLE_GOOGLEBENCH` to ON, `BUILD_TESTING` is automatically enabled. After building and installing, performance tests can be executed with `ctest`.

5.7 Build with CUDA support

5.7.1 Extra prerequisites

- CUDA-Toolkit

To build NVIDIA GPU support for ABACUS, define `USE_CUDA` flag. You can also specify path to local installation of CUDA Toolkit by setting `CMAKE_CUDA_COMPILER` flags.

```
cmake -B build -DUSE_CUDA=1 -DCMAKE_CUDA_COMPILER=${path to cuda toolkit}/bin/nvcc
```

5.8 Build math library from source

> Note: This flag is **enabled by default**. It will get better performance than the standard implementation on `gcc` and `clang`. But it **will be disabled** when using `Intel Compiler` since the math functions will get wrong results and the performance is also unexpectly poor.

To build math functions from source code, instead of using c++ standard implementation, define `USE_ABACUS_LIBM` flag.

Currently supported math functions: `sin`, `cos`, `sincos`, `exp`, `cexp`

```
cmake -B build -DUSE_ABACUS_LIBM=1
```

5.9 Build ABACUS with make

> Note: We suggest using CMake to configure and compile.

To compile the ABACUS program using legacy `make`, users need to specify the location of the compilers, headers and libraries in `source/Makefile.vars`:

```
# This is the Makefile of ABACUS API
#=======================================================================
# Users set
#=======================================================================
CXX = mpiicpc
# mpiicpc:   compile intel parallel version
```

(continues on next page)

(continued from previous page)

```
# icpc:      compile intel sequential version
# make: ELPA_DIR, ELPA_INCLUDE_DIR, CEREAL_DIR must also be set.
# make pw: nothing need to be set except LIBXC_DIR
#
# mpicxx:    compile gnu parallel version
# g++:       compile gnu sequential version
# make: FFTW_DIR, OPENBLAS_LIB_DIR, SCALAPACK_LIB_DIR, ELPA_DIR, ELPA_INCLUDE_DIR,␣
↪CEREAL_DIR must also be set.
# make pw: FFTW_DIR, OPENBLAS_LIB_DIR must be set.

# GPU = OFF  #We do not support GPU yet
# OFF:  do not use GPU
# CUDA: use CUDA
#=========================================================================



#--------------------  FOR INTEL COMPILER  ----------------------------
## ELPA_DIR          should contain an include folder and lib/libelpa.a
## CEREAL_DIR        should contain an include folder.
#-------------------------------------------------------------------------

ELPA_DIR      = /usr/local/include/elpa-2021.05.002
ELPA_INCLUDE_DIR = ${ELPA_DIR}/elpa

CEREAL_DIR    = /usr/local/include/cereal


##--------------------  FOR GNU COMPILER  -----------------------------
## FFTW_DIR          should contain lib/libfftw3.a.
## OPENBLAS_LIB_DIR  should contain libopenblas.a.
## SCALAPACK_LIB_DIR should contain libscalapack.a
## All three above will only be used when CXX=mpicxx or g++
## ELPA_DIR          should contain an include folder and lib/libelpa.a
## CEREAL_DIR        should contain an include folder.
##-------------------------------------------------------------------------

# FFTW_DIR = /public/soft/fftw_3.3.8
# OPENBLAS_LIB_DIR   = /public/soft/openblas/lib
# SCALAPACK_LIB_DIR  = /public/soft/openblas/lib

# ELPA_DIR      = /public/soft/elpa_21.05.002
# ELPA_INCLUDE_DIR = ${ELPA_DIR}/include/elpa-2021.05.002

# CEREAL_DIR    = /public/soft/cereal


##--------------------  OPTIONAL LIBS  ---------------------------------
## To use DEEPKS: set LIBTORCH_DIR and LIBNPY_DIR
## To use LIBXC:  set LIBXC_DIR which contains include and lib/libxc.a (>5.1.7)
## To use DeePMD: set DeePMD_DIR and TensorFlow_DIR
## To use LibRI:  set LIBRI_DIR and LIBCOMM_DIR
##-------------------------------------------------------------------------

# LIBTORCH_DIR  = /usr/local
# LIBNPY_DIR    = /usr/local
```

(continues on next page)

(continued from previous page)

```
# LIBXC_DIR                    = /public/soft/libxc

# DeePMD_DIR = ${deepmd_root}
# TensorFlow_DIR = ${tensorflow_root}

# LIBRI_DIR     = /public/software/LibRI
# LIBCOMM_DIR   = /public/software/LibComm

##----------------------------------------------------------------------
# NP = 14 # It is not supported. use make -j14 or make -j to parallelly compile
# DEBUG = OFF
# Only for developers
# ON:   use gnu compiler and check segmental defaults
# OFF:  nothing
#======================================================================
```

For example, below is a case where the Intel C++ compiler, Intel MPI and CEREAL are used, along with Intel MKL library. The file Makefile.vars can be set as follows:

```
CXX = mpiicpc #(or CXX = icpc)
ELPA_DIR      = /public/soft/elpa_21.05.002
ELPA_INCLUDE_DIR = ${ELPA_DIR}/include/elpa-2021.05.002
CEREAL_DIR    = /public/soft/cereal
```

When `CXX=mpiicpc`, a parallel version will be compiled. When `CXX=icpc`, a sequential version will be compiled.

Another example is where the Gnu C++ compiler, MPICH, OPENBLAS, ScaLAPACK, ELPA and CEREAL are used:

```
CXX = mpicxx/g++
FFTW_DIR = /public/soft/fftw_3.3.8
OPENBLAS_LIB_DIR   = /public/soft/openblas/lib
SCALAPACK_LIB_DIR  = /public/soft/openblas/lib
ELPA_DIR      = /public/soft/elpa_21.05.002
ELPA_INCLUDE_DIR = ${ELPA_DIR}/include/elpa-2021.05.002
CEREAL_DIR    = /public/soft/cereal
```

When `CXX=mpicxx`, a parallel version will be compiled. When `CXX=g++`, a sequential version will be compiled.

Except modifying `Makefile.vars`, you can also directly use

```
make CXX=mpiicpc ELPA_DIR=/public/soft/elpa_21.05.002 \
ELPA_INCLUDE_DIR=${ELPA_DIR}/include/elpa-2021.05.002 \
CEREAL_DIR=/public/soft/cereal
```

ABACUS now support full version and pw version. Use `make` or `make abacus` to compile full version which supports LCAO calculations. Use `make pw` to compile pw version which only supports pw calculations. For pw version, `make pw CXX=mpiicpc`, you do not need to provide any libs. For `make pw CXX=mpicxx`, you need provide `FFTW_DIR` and `OPENBLAS_LIB_DIR`.

Besides, libxc and deepks are optional libs to compile abacus. They will be used when `LIBXC_DIR` is defined like

```
LIBXC_DIR = /public/soft/libxc
```

or `LIBTORCH_DIR` and `LIBNPY_DIR` like

```
LIBTORCH_DIR  = /usr/local
LIBNPY_DIR    = /usr/local
```

After modifying the `Makefile.vars` file, execute `make` or `make -j12` to build the program.

After the compilation finishes without error messages (except perhaps for some warnings), an executable program `ABACUS.mpi` will be created in directory `bin/`.

5.9.1 Add Libxc Support

The program compiled using the above instructions do not link with LIBXC and use exchange-correlation functionals as written in the ABACUS program. However, for some functionals (such as HSE hybrid functional), LIBXC is required.

To compile ABACUS with LIBXC, you need to define `LIBXC_DIR` in the file `Makefile.vars` or use

```
make LIBXC_DIR=/pulic/soft/libxc
```

directly.

5.9.2 Add DeePKS Support

To compile ABACUS with DEEPKS, you need to define `LIBTORCH_DIR` and `LIBNPY_DIR` in the file `Makefile.vars` or use

```
make LIBTORCH_DIR=/opt/libtorch/ LIBNPY_DIR=/opt/libnpy/
```

directly.

5.9.3 Add DeePMD-kit Support

> Note: This part is only required if you want to load a trained DeeP Potential and run molecular dynamics with that. To train the DeeP Potential with DP-GEN, no extra prerequisite is needed and please refer to this page for ABACUS interface with DP-GEN.

To compile ABACUS with DeePMD-kit, you need to define `DeePMD_DIR` and `TensorFlow_DIR` in the file `Makefile.vars` or use

```
make DeePMD_DIR=~/deepmd-kit TensorFlow_DIR=~/tensorflow
```

directly.

> `deepmd_c`/`deepmd_cc` and `tensorflow_cc` libraries would be called according to `DeePMD_DIR` and `TensorFlow_DIR`, which is showed in detail in this page.

5.9.4 Add LibRI Support

To use new EXX, you need two libraries: LibRI and LibComm and need to define `LIBRI_DIR` and `LIBCOMM_DIR` in the file `Makefile.vars` or use

```
make LIBRI_DIR=/public/software/LibRI LIBCOMM_DIR=/public/software/LibComm
```

directly.

CHAPTER

SIX

RUNNING SCF

6.1 Initializing SCF

Good initializing would abate the number of iteration steps in SCF. Charge density should be initialed for constructing the initial hamiltonian operator.

In PW basis, wavefunction should be initialized for iterate diagonalization method. In LCAO basis, wavefunction can be read to calculate initial charge density. The wavefunction itself does not have to be initialized.

6.1.1 Charge Density

`init_chg` is used for choosing the method of charge density initialization.

- `atomic` : initial charge density by atomic charge density from pseudopotential file under keyword `PP_RHOATOM`
- `file` : initial charge density from files produced by previous calculations with *`out_chg 1`*.

6.1.2 Wave function

`init_wfc` is used for choosing the method of wavefunction coefficient initialization.

When `basis_type=pw`, setting of `random` and `atomic` are supported. Atomic wave function is read from pseudopotential file under keyword `PP_PSWFC`, if setting is `atomic` and number of band of atomic wavefunction less than `nbands` in INPUT file, the extra bands will be initialed by random.

When `basis_type=lcao`, we further support reading of initial wavefunction by setting `init_wfc` to `file`. In LCAO code, wave function is used to initialize density matrix and real-space charge density. For such purpose, a file containing wavefunction must be prepared. Such files can be generated from previous calculations with *`out_wfc_lcao 1`*.

6.2 Constructing the Hamiltonian

6.2.1 Exchange-Correlation Functionals

In our package, the XC functional can be set explicitly using the `dft_functional` keyword in `INPUT` file. If `dft_functional` is not specified, ABACUS will use the xc functional indicated in the pseudopotential file.

Several common functionals are implemented in ABACUS, such as PZ and PBE. Users can check out this file for a complete list of functionals implemented in ABACUS. Furthermore, if ABACUS is compiled with LIBXC, we also support all the LDA, GGA and meta-GGA functionals provided therein.

Here, we use a simple example calculation for illustration.

1. **Default setting:**

   In the original `INPUT` file, there is no specification of the `dft_functional` keyword. As a result, we use the default option, that is to use the xc functional in the pseudopotential file, `Si.pz-vbc.UPF`. We can take a look at the first few lines of the `<PP_HEADER>` section from the pseudopotential file:

   ```
   <PP_HEADER>
   0                    Version Number
   Si                    Element
   NC                   Norm - Conserving pseudopotential
       F                   Nonlinear Core Correction
   SLA  PZ   NOGX NOGC   PZ   Exchange-Correlation functional
   ```

   From the line commented 'Exchange-Correlation functional', we see that this pseudopotential is generated using PZ functional. As a result, if we run ABACUS with the original setting, PZ functional will be used.

   > Note : for systems with multiple elements, if no `dft_functional` is specified, users should make sure that all pseudopotentials are using the same functional. Otherwise, the type of xc functional should be specified explicitly.

2. **Using PBE**

   On the other hand, users might also explicitly specify the xc functional through `dft_functional` parameter. For example, to use PBE functional, add the following line to `INPUT` file and rerun the calculation:

   ```
   dft_functional PBE
   ```

3. **More functionals from LIBXC**

   ABACUS has its own implementation of the PBE functional as well as a few others, but our list is far from comprehensive. However, if ABACUS is compiled with LIBXC, we also support all the LDA, GGA and meta-GGA functionals provided therein.

   For this part, users should compile the ABACUS code with LIBXC linked (version 5.1.7 or higher).

   To use SCAN functional, make the following modification to the `INPUT` file:

   ```
   dft_functional SCAN
   ```

   Note that in the case of PBE and SCAN, we are using 'short-hand' names to represent the entire functional, which is made up of individual exchange and correlation components. A complete list of 'short-hand' expressions supported by ABACUS can be found in source code.

   Apart from the 'short-hand' names, ABACUS also allow supplying exchange-correlation functionals as combinations of LIBXC keywords for functional components, joined by plus sign, for example, setting:

   ```
   dft_functional LDA_X_YUKAWA+LDA_C_1D_CSC
   ```

   means we are using the short-range Yukawa attenuated exchange along with the Casula, Sorella & Senatore LDA correlation functional.

   The list of LIBXC keywords can be found on its website.

4. **Temperature-dependent functional**

   In ABACUS, we provide temperature-dependent functionals through LIBXC. For such functionals, the keyword `xc_temperature` (unit is Rydberg) is used to specify the temperature, such as the following:

   ```
   dft_functional LDA_XC_CORRKSDT
   xc_temperature 10
   ```

5. **Hybrid functional**

   ABACUS supports functionals with exact Hartree-Fock exchange in LCAO basis set only. The old INPUT parameter exx_hybrid_type for hybrid functionals has been absorbed into `dft_functional`. Options are `hf` (pure Hartree-Fock), `pbe0`(PBE0), `hse`, and `scan0`(SCAN0) (Note: in order to use HSE or SCAN0 functional, LIBXC is required). Note also that only HSE has been tested while other hybrid functionals have NOT been fully tested yet, and the maximum parallel cpus for running exx is N^4, with N being the number of atoms.

   More information on the hybrid functional can be found from the section *Exact Exchange* in the list of input variables for more information.

   An example HSE calculation is provided in this directory. Apart from the input files (`INPUT`, `STRU`, `KPT`), we further provide two files: running_scf.log_ref and log_ref, which contains reference for running_scf.log and standard output from the program, respectively.

6.2.2 DFT+*U*

Conventional functionals, e.g., L(S)DA and GGAs, encounter failures in strongly correlated systems, usually characterized by partially filled *d*/*f* shells. These include transition metals ™ and their oxides, rare-earth compounds, and actinides, to name a few, where L(S)DA/GGAs typically yield quantitatively or even qualitatively wrong results. To address this failure, an efficient and successful method named DFT+*U*, which inherits the efficiency of L(S)DA/GGA but gains the strength of the Hubbard model in describing the physics of strongly correlatedsystems, has been developed.

Now the DFT+*U* method is accessible in ABACUS. The details of the DFT+*U* method could be found in this paper. It should be noted that the DFT+*U* works only within the NAO scheme, which means that the value of the keyword `basis_type` must be lcao when DFT+*U* is called. To turn on DFT+*U*, users need to set the value of the `dft_plus_u` keyword in the `INPUT` file to be 1. All relevant parmeters used in DFT+*U* calculations are listed in the *DFT+U correction* part of the *list of keywords*.

Examples of DFT+*U* calculations are provided in this directory.

6.3 Solving the Hamiltonian

6.3.1 Explicit Diagonalization

Method of explicit solving KS-equation can be chosen by variable “ks_solver” in INPUT file.

When “basis_type = pw”, `ks_solver` can be `cg`, `bpcg` or `dav`. The `bpcg` method only supports K-point parallelism currently. The default setting `cg` is recommended, which is band-by-band conjugate gradient diagonalization method. There is a large probability that the use of setting of `dav` , which is block Davidson diagonalization method, can be tried to improve performance.

When “basis_type = lcao”, `ks_solver` can be `genelpa` or `scalapack_gvx`. The default setting `genelpa` is recommended, which is based on ELPA (EIGENVALUE SOLVERS FOR PETAFLOP APPLICATIONS) (https://elpa.mpcdf.mpg.de/) and the kernel is auto choosed by GENELPA(https://github.com/pplab/GenELPA), usually faster than the setting of “scalapack_gvx”, which is based on ScaLAPACK(Scalable Linear Algebra PACKage)

6.3.2 Stochasic DFT

We support stochastic DFT calculation (SDFT) or mixed stochastic-deterministic DFT (MDFT) with plane-wave basis [Phys. Rev. B 106, 125132 (2022)]. Different from traditional KSDFT with the explicit diagonalization method, SDFT and MDFT calculate physical quantities with trace of the corresponding operators. The advantages of SDFT and MDFT compared to the traditional KSDFT are the ability to simulate larger sizes and higher temperatures. In our package, SDFT and MDFT can be used by setting the `esolver_type` parameter to `sdft` for SCF calculations or MD calculations. To start with, you can refer to two examples and an explanation of the *input variables*.

When we have a hamiltonian, the electronic density can be calculated with:

$\rho(\mathbf{r}) = \mathrm{Tr}[f(\hat{H})\mathbf{rr}]$,

where the Fermi-Dirac function $f(\hat{H}) = \frac{1}{1+\exp(\frac{\hat{H}-\mu}{kT})}$ and it can be calculated with the Chebyshev expansion. Here we only support the "fd" or "fermi-dirac" `smearing_method`, the parameter `smearing_sigma` is equal the temperature T (in Ry) and `nche_sto` represents the order of the expansion.

For physical quantities represented by operator $\hat{O}$, SDFT calculates its trace with:

$\mathrm{Tr}[\hat{O}] = \sum_{i=1}^{N_\chi} \chi_i \hat{O} \chi_i$,

while MDFT calculates the trace as:

$\mathrm{Tr}[\hat{O}] = \sum_{n=1}^{N_\phi} \phi_n \hat{O} \phi_n + \sum_{i=1}^{N_\chi} \tilde{\chi}_i \hat{O} \tilde{\chi}_i$,

where $\{\tilde{\chi}_i\}$ are obtaiend by projecting stochastic orbitals onto the subspace orthogonal to KS orbitals $\{\phi_n\}$:

$\tilde{\chi}_i = \chi_i - \sum_{n=1}^{N_\phi} \phi_n |\chi_i \phi_n$.

Here the number of KS orbitals N_ϕ is controlled by the parameter `nbands` while the number of stochastic orbitals N_χ is controlled by `nbands_sto`.

Besides, although SDFT does not diagonalize the hamiltonian, it can also caluclate DOS and electronic conductivities with parameters `out_dos` and `cal_cond` separately.

6.4 Converging SCF

As in any non-linear systems, numerical instabilities during SCF iterations may lead to nonconvergence. In ABACUS, we offer the following options to facilitate SCF convergence.

6.4.1 Charge Mixing

In ABACUS, KS-DFT is solved by self-consistent field (SCF) iteration method. By mixing the electron density with that obtained from previous steps, numerical instabilities can be ameliorated while also accelerating convergence. ABACUS offers several mixing schemes, and users may make a selection by adjusting the *mixing_type* keyword in INPUT file.

For each of the mixing types, we also provide variables for controlling relevant parameters, including `mixing_ndim`, `mixing_type`, `mixing_beta`, `mixing_gg0`, `mixing_beta_mag`, `mixing_gg0_mag`, `mixing_gg0_min`, `mixing_angle`.

`mixing_ndim` is the mixing dimensions in DIIS (broyden or pulay) mixing. Gerenally, a larger `mixing_ndim` leads to a better convergence. the default choice `mixing_ndim=8` should work fine in most cases. For `mixing_type`, the default choice is `broyden`, which is slightly better than `Pulay` typically. Besides that, a large `mixing_beta` means a larger change in electron density for each SCF step. For well-behaved systems, a larger `mixing_beta` leads to faster convergence. However, for some difficult cases, a smaller `mixing_beta` is preferred to avoid numerical instabilities.

For most isolated systems, Kerker preconditioning is unnecessary. You can turn off it by setting `mixing_gg0 0.0` to get a faster convergence.

For non-spin-polarized calculations, the default choices usually achieve convergence. If convergence issue arises in metallic systems, you can try different value of Kerker preconditioning *mixing_gg0* and *mixing_gg0_min*, and try to reduce `mixing_beta`, which is 0.8 defaultly for `nspin=1`.

For magnetic calculations, `mixing_beta_mag` and `mixing_gg0_mag` are activated. Considering collinear calculations, you can rely on the default value for most cases. If convergence issue arises, you can try to reduce `mixing_beta` and `mixing_beta_mag` together. For non-collinear calculations, tradtional broyden usually works, especially for a given magnetic configuration. If one is not interested in the energies of a given magnetic configuration but wants to determine the ground state by relaxing the magnetic moments' directions, the standard Broyden mixing algorithm sometimes fails to find the correct magnetic configuration. If so, we can set *mixing_angle=1.0*, which is a promising mixing method proposed by J. Phys. Soc. Jpn. 82 (2013) 114706.

An example showcasing different charge mixing methods can be found in our repository. Four INPUT files are provided, with description given in README.

As for DFT+U calculations, where the hamiltonian is not only dependent on charge density, but also dependent on density matrix. You can try `mixing_restart>0` and `mixing_dmr=1` to improve convergence. For case extremely hard to converge, you can use so-called U-Ramping method by setting a finite positive `uramping` with `mixing_restart>0` and `mixing_dmr=1`.

6.4.2 Smearing

Thermal smearing is an efficient tool for accelerating SCF convergence by allowing fractional occupation of molecular orbitals near the band edge. It is important for metallic systems.

In ABACUS, we provide a few smearing methods, which can be controlled using the keyword *smearing_method*. We also provide keyword `smearing_sigma` or `smearing_sigma_temp` to control the energy range of smearing. A larger value of smearing sigma leads to a more diffused occupation curve.

> Note : The two keywords `smearing_sigma` and `smearing_sigma_temp` should not be used concurrently.

We provide an example showing the importance of smearing in our repository. Two INPUT fiels rae provided, with description given in README.

6.5 Accelerating the Calculation

In ABACUS, we provide a few methods for accelerating the calculation. The parameters are usually set as default for calculations where there is not extreme concern for efficiency, as some of them may produce numerical issues under certain circumstances. In short, methods in this section should be used with care. It is better to calibrate the results against the default setting.

6.5.1 K-point Parallelization

In ABACUS, we offer k-point parallelization for calculations with PW basis, which should increase the efficiency when a large k-point mesh is used.

To use k-point parallelization, users may set keyword *kpar* to be larger than 1.

> Note: It has been observed that k-point parallelization cannot work in conjunction with Davidson diagonalization.

6.5.2 K-point Symmetry

Inclusion of k-point symmetry helps increasing the efficiency of calculations by reducing the effective number of k-points used. To turn on k-point symmetry, users may set keyword *symmetry* to be 1.

> Note: In ABACUS we only support point-group symmetry but not space-group symmetry.

6.5.3 Accelerating Grid Integration

For LCAO calculation, the matrix elements of the local potential is evaluated using grid integration. In grid integration, we group real-space FFT grid points into boxes of dimension bx * by * bz, and then proceed with the boxes as the basis unit of calculation.

Setting *bx, by, bz* to be values other than default might help with the efficiency of grid integration.

> Note: the choice of bx, by, bz should be integer factors of the dimension of the real space FFT grid in each direction.

6.5.4 Low Dimension Materials

In grid integration, we chose to parallelize the grid points along the z direction. Therefore, when using LCAO calculation for low dimension materials, it is recommended to put the material more evenly in z direction to avoid imbalanced workload on different MPI threads.

Namely, when calculating 2D materials, it is better to put the material in xz or yz direction; while for 1D materials, it is better to align the material with the z direction.

6.6 SCF in Complex Environments

6.6.1 Implicit Solvation Model

Solid-liquid interfaces are ubiquitous in nature and frequently encountered and employed in materials simulation. The solvation effect should be taken into account in first-principles calculations of such systems so as to obtain accurate predictions.

Implicit solvation model is a well-developed method to deal with solvation effects, which has been widely used in finite and periodic systems. This approach treats the solvent as a continuous medium instead of individual "explicit" solvent molecules, which means that the solute embedded in an implicit solvent, and the average over the solvent degrees of freedom becomes implicit in the properties of the solvent bath. Compared to the "explicit" method, such implicit solvation model can provide qualitatively correct results with much less computational cost, which is particularly suited for large and complex systems. The implicit solvation model implemented in ABACUS follows the methodology developed by Mathew, Sundararaman, Letchworth-Weaver, Arias, and Hennig in 2014.

Input parameters that control the implicit solvation model are listed as follows with detailed explaination and recommended values provided on this webpage:

```
INPUT_PARAMETERS
imp_sol                 1
eb_k                    80
tau                     0.000010798
sigma_k                 0.6
nc_k                    0.00037
```

Example of running DFT calculation with the implicit solvation model is provided in this directory.

6.6.2 External Electric Field

A saw-like potential simulating an electric field can be added to the bare ionic potential, which is a simplified simulation to the field-effect measurements, in which the system is separated from the gate electrode by a dielectric such as silicon oxide.

Whether to apply the external field is controlled via the keyword `efield_flag` in `INPUT` (setting to 1 to turn on the field). Related keywords that control the external field are listed as follows with detailed explaination provided here:

```
INPUT_PARAMETERS
efield_flag        1
efield_dir         2
efield_pos_max     0.5
efield_pos_dec     0.1
efield_amp         0.001
```

Example of running DFT calculation with added external electric field is provided in this directory.

6.6.3 Dipole Correction

A dipole correction can be added to the bare ionic potential, which can compensate for the artificial dipole field within the context of a periodic supercell calculation. The dipole correction implemented in ABACUS follows the methodology proposed by Bengtsson in 1999. This correction must be used ONLY in a slab geometry, for surface calculations, with the discontinuity FALLING IN THE EMPTY SPACE. Note that the common input parameters shared between the external electric field and dipole correction, with detailed explaination provided here. The following keywords settings add dipole correction only without applying any external electric field:

```
INPUT_PARAMETERS
efield_flag        1
dip_cor_flag       1
efield_dir         2
efield_pos_max     0.5
efield_pos_dec     0.1
efield_amp         0
```

While The external electric field and dipole correction can also be added together to the bare ionic potential as follows:

```
INPUT_PARAMETERS
efield_flag        1
dip_cor_flag       1
efield_dir         2
efield_pos_max     0.5
efield_pos_dec     0.1
efield_amp         0.001
```

Examples of running DFT calculations with dipole correction are provided in this directory. There are two input files, where `INPUT1` considers only the dipole correction without no applied external field, while `INPUT2` considers the dipole correction under an applied external field.

To run any of the two cases, users may enter the directory, copy the corresponding input file to `INPUT`, and run ABACUS.

6.6.4 Compensating Charge

Modeling a constant-potential electronchemcial surface reaction requires adjustment of electron numbers in a simulation cell. At the mean time, we need to maintain the supercell's neutrality due to the periodic boundary condition. A distribution of compensating charge thus needs to be implemented in the vacuum region of surface models when extra electrons are added/extracted from the system.

The compensating charge implemented in ABACUS follows the methodology developed by Brumme, Calandra, and Mauri in 2014. Input parameters that control the compensating charge are listed as follows with detailed explaination provided here:

```
INPUT_PARAMETERS
gate_field          1
efield_dir          2
zgate               0.5
block               1
block_down          0.45
block_up            0.55
block_height        0.1
```

Example of running DFT calculation with the compensating charge is provided in this directory.

6.6.5 Van-der-Waals Correction

Conventional DFT functionals often suffer from an inadequate treatment of long-range dispersion, or Van der Waals (VdW) interactions. In order to describe materials where VdW interactions are prominent, one simple and popular approach is to add a Lennard-Jones type term. The resulting VdW-corrected DFT has been proved to be a very effective method for description of both short-range chemical bonding and long-range dispersive interactions.

Currently ABACUS provides three Grimme DFT-D methods, including D2, D3(0) and D3(BJ), to describe Van der Waals interactions. Among them, the D3 method has been implemented in ABACUS based on the dftd3 program written by Stefan Grimme, Stephan Ehrlich and Helge Krieg.

To use VdW-correction, users need to supply value to the `vdw_method` keyword in the `INPUT` file:

- (Default) none: no VdW correction
- d2: DFT-D2 method
- d3_0: DFT-D3(0) method
- d3_bj: DFT-D3(BJ) method

Furthermore, ABACUS also provides a *list of keywords* to control relevant parmeters used in calculating the VdW correction, such as the scale factor (s6) term. Recommended values of such parameters can be found on the webpage. The default values of the parameters in ABACUS are set to be the recommended values for PBE.

Examples of VdW-corrected DFT calculations are provided in this directory. There are two input files, where `INPUT1` shows how to apply D2 correction with user-specified C_6 parameter, and `INPUT2` shows how to apply D3(BJ) correction with default VdW parameters.

To run any of the two cases, users may enter the directory, copy the corresponding input file to `INPUT`, and run ABACUS.

6.7 Spin-polarization and SOC

6.7.1 Non-spin-polarized Calculations

Setting of **"nspin 1"** in INPUT file means calculation with non-polarized spin. In this case, electrons with spin up and spin down have same occupations at every energy states, weights of bands per k point would be double.

6.7.2 Collinear Spin Polarized Calculations

Setting of **"nspin 2"** in INPUT file means calculation with polarized spin along z-axis. In this case, electrons with spin up and spin down will be calculated respectively, number of k points would be doubled. Potential of electron and charge density will separate to spin-up case and spin-down case.

Magnetic moment Settings in *STRU files* are not ignored until **"nspin 2"** is set in INPUT file

When **"nspin 2"** is set, the screen output file will contain magnetic moment information. e.g.

```
ITER   TMAG      AMAG      ETOT(eV)        EDIFF(eV)       DRHO        TIME(s)
GE1    4.16e+00  4.36e+00  -6.440173e+03   0.000000e+00    6.516e-02   1.973e+01
```

where "TMAG" refers to total magnetization and "AMAG" refers to average magnetization. For more detailed orbital magnetic moment information, please use *Mulliken charge analysis*.

Constraint DFT for collinear spin polarized calculations

For some special need, there are two method to constrain electronic spin.

1. **"ocp"** and **"ocp_set"** If **"ocp=1"** and **"ocp_set"** is set in INPUT file, the occupations of states would be fixed by **"ocp_set"**, this method is often used for excited states calculation. Be careful that: when **"nspin=1"**, spin-up and spin-down electrons will both be set, and when **"nspin=2"**, you should set spin-up and spin-down respectively.
2. **"nupdown"** If **"nupdown"** is set to non-zero, number of spin-up and spin-down electrons will be fixed, and Fermi energy level will split to E_Fermi_up and E_Fermi_down. By the way, total magnetization will also be fixed, and will be the value of **"nupdown"**.
3. DeltaSpin The `DeltaSpin` function as proposed by Zefeng Cai and Ben Xu, et al. arXiv:2208.04551v6 has been implemented in ABACUS in the LCAO basis for the `nspin 2` case. In order to use this function, the following parameters are needed to be set in the input file, for example:

```
#deltaspin
sc_mag_switch                   1
decay_grad_switch               0
sc_thr                          1e-7
nsc                             150
nsc_min                         2
sc_file                         sc.json
alpha_trial                     0.01
sccut                           3
```

The explanation of each input paramters has been explained in the *Noncollinear Spin Polarized Calculations* section.

An example of the sc_file json file is shown below:

```
[
    {
        "element": "Fe",
        "itype": 0,
        "ScDecayGrad": 0.9,
        "ScAtomData": [
            {
                "index": 0,
                "lambda": 0.0,
                "target_mag": 2.0,
                "constrain": 1
            },
            {
                "index": 1,
                "lambda": 0,
                "target_mag": 2.0,
                "constrain": 1
            }
        ]
    }
]
```

Please refer the *Noncollinear Spin Polarized Calculations* section for the explanation of each input paramters. The difference is that `lambda`, `target_mag`, and `constrain` are scalars instead of vectors. Simple examples are provided in the `abacus-develop/examples/spin_polarized` directory.

6.7.3 Noncollinear Spin Polarized Calculations

The spin non-collinear polarization calculation corresponds to setting **"noncolin 1"**, in which case the coupling between spin up and spin down will be taken into account. In this case, nspin is automatically set to 4, which is usually not required to be specified manually. The weight of each band will not change, but the number of occupied states will be double. If the nbands parameter is set manually, it is generally set to twice what it would be when nspin<4.

In general, non-collinear magnetic moment settings are often used in calculations considering *SOC effects*. When **"lspinorb 1"** in INPUT file, "nspin" is also automatically set to 4. Note: different settings for "noncolin" and "lspinorb" correspond to different calculations:

- noncolin=0 lspinorb=0 nspin<4 : Non-collinear magnetic moments and SOC effects are not considered.
- noncolin=0 lspinorb=0 nspin=4 : Actualy same as the above setting, but the calculation will be larger. So the setting is not recommended.
- noncolin=1 lspinorb=0 : Non-collinear magnetic moments are considered but SOC effects are not considered
- noncolin=0 lspinorb=1 : The SOC effect is considered but the magnetic moment is limited to the Z direction
- noncolin=1 lspinorb=1 : The SOC effect and non-collinear magnetic moment are both calculated.

Constraint Spin functionality for noncollinear spin polarized calculations

The `DeltaSpin` function as proposed by Zefeng Cai and Ben Xu, et al. arXiv:2208.04551v6 has been implemented in ABACUS in the LCAO basis. In order to use this function, the following parameters are needed to be set in the input file, for example:

```
#deltaspin
sc_mag_switch                   1
decay_grad_switch               1
sc_thr                          1e-7
nsc                             150
nsc_min                         2
sc_file                         sc.json
alpha_trial                     0.01
sccut                           3
```

`sc_mag_switch` is the switch of deltaspin functionality; `decay_grad_switch` is the switch of decay gradient method; `sc_thr` is the threshold of the spin constraint atomic magnetic moment in unit of Bohr Mag (\muB); `nsc` is the number of self-consistent iterations; `nsc_min` is the minimum number of self-consistent iterations; `sc_file` is the file name of the spin constraint parameters; `alpha_trial` is the initial trial step size for lambda in eV/uB^2; `sccut` restriction of step size in eV/uB.

An example of the sc_file json file is shown below:

```
[
    {
        "element": "Fe",
        "itype": 0,
        "ScDecayGrad": 0.9,
        "ScAtomData": [
            {
                "index": 0,
                "lambda": [0, 0, 0],
                "target_mag": [2.0, 0.0, 0.0],
                "constrain": [1,1,1]
            },
            {
                "index": 1,
                "lambda": [0, 0, 0],
                "target_mag_val": 2.0,
                "target_mag_angle1": 80.0,
                "target_mag_angle2": 0.0,
                "constrain": [1,1,1]
            }
        ]
    }
]
```

The sc_file json file is a list of elemental data in total. For each element, the user should specify its name, the `itype` parameter should be in accord with `STRU` file and start from 0. `ScDecayGrad` is a parameter for each element in unit of (uB^2/eV), this parameter needs to be determined for different element, for example, 0.9 uB^2/eV is an appropriate value for BCC-Fe according to Zefeng Cai's tests. `ScAtomData` specifies spin constraining parameters for each atom, the `index` starts from 0 and corresponds atomic order in the `STRU` file. `lambda` is a 3d vector for each atom, and it is recommended to set to [0.0, 0.0, 0.0] for all atoms. Users have two optional choices to set the target magnetic moments for each atom, i.e., by a 3d vector or by angles. If the `target_mag` is set, the `target_mag_val` and `target_mag_angle1` and `target_mag_angle2` will be ignored. The `target_mag` is a 3d vector in unit of Bohr Mag (\muB), and the `target_mag_val` is a scalar value in unit of Bohr Mag (\muB), `target_mag_angle1`

and target_mag_angle2 are two angles in unit of degree. The constrain is a 3d vector, if the corresponding element is set to 1, the corresponding component of the magnetic moment will be constrained, otherwise, it will be free. Note that the initial atomic magnetic moments are still set in the STRU file. Simple examples are provided in the abacus-develop/examples/noncollinear directory. One should set noncolliear to 1 to run the DeltaSpin function, lspinorb=1 is not mandatory, but it is recommended to set to 1 to get more accurate results.

6.7.4 For the continuation job

- Continuation job for "nspin 1" need file "SPIN1_CHG.cube" which is generated by setting "out_chg=1" in task before. By setting "init_chg file" in new job's INPUT file, charge density will start from file but not atomic.
- Continuation job for "nspin 2" need files "SPIN1_CHG.cube" and "SPIN2_CHG.cube" which are generated by "out_chg 1" with "nspin 2", and refer to spin-up and spin-down charge densities respectively. It should be note that reading "SPIN1_CHG.cube" only for the continuation target magnetic moment job is not supported now.
- Continuation job for "nspin 4" need files "SPIN%s_CHG.cube", where %s in {1,2,3,4}, which are generated by "out_chg 1" with any variable setting leading to 'nspin'=4, and refer to charge densities in Pauli spin matrixes. It should be note that reading charge density files printing by 'nspin'=2 case is supported, which means only σ_{tot} and σ_z are read.

6.8 SOC Effects

6.8.1 SOC

lspinorb is used for control whether or not SOC(spin-orbit coupling) effects should be considered.

Both basis_type=pw and basis_type=lcao support scf and nscf calculation with SOC effects.

Atomic forces and cell stresses can not be calculated with SOC effects yet.

6.8.2 Pseudopotentials and Numerical Atomic Orbitals

For Norm-Conserving pseudopotentials, there are differences between SOC version and non-SOC version.

Please check your pseudopotential files before calculating. In PP_HEADER part, keyword has_so=1 and relativistic="full" refer to SOC effects have been considered, which would lead to different PP_NONLOCAL and PP_PSWFC parts. Please be careful that relativistic="full" version can be used for SOC or non-SOC calculation, but relativistic="scalar" version only can be used for non-SOC calculation. When full-relativistic pseudopotential is used for non-SOC calculation, ABACUS will automatically transform it to scalar-relativistic version.

Numerical atomic orbitals in ABACUS are unrelated with spin, and same orbital file can be used for SOC and non-SOC calculation.

6.8.3 Partial-relativistic SOC Effect

Sometimes, for some real materials, both scalar-relativistic and full-relativistic can not describe the exact spin-orbit coupling. Artificial modulation can help for these cases.

`soc_lambda`, which has value range [0.0, 1.0] , is used for modulate SOC effect. In particular, `soc_lambda 0.0` refers to scalar-relativistic case and `soc_lambda 1.0` refers to full-relativistic case.

CHAPTER

SEVEN

BASIS SET AND PSEUDOPOTENTIALS

7.1 Basis Set

ABACUS supports both PW and LCAO basis set, controlled by keyword *basis_type* in INPUT file.

The default value of basis_type is pw. The size of pw basis set is controlled by imposing an upper bound for the *kinetic energy cutoff* of the plane wave.

When choosing lcao basis set, users need to prepare a set of atomic orbitals. Such files may be downloaded from the official website. For more information, also check the `NUMERICAL_ORBITAL` section in the specification of the *STRU file*.

The sequence of orbitals in lcao basis set is as follows. First, all the orbitals belonging to one particular atom are put together. These atom orbitals are arranged as the atom order specified in the STRU file. Then, the orbitals of each atom are arranged according to the orbital files. If the orbital file says that the number of s□p□d…orbitals is n_s□n_p□n_d…then the orbitals are aligned as first n_s s orbitals, then n_p p orbitals, and then n_d d orbitals…Last, the angular part of each orbital is real spherical harmonic function. They are aligned as Y00, Y10, Y11, Y1-1, Y20, Y21, Y2-1, Y22, Y2-2, which is s,p_z,p_x,p_y,d_{z^2},d_{xz},d_{yz},$d_{x^2-y^2}$,d_{xy}. The corresponding formula can be seen in Table of spherical harmonics - Wikipedia. Note that these formula lack of the Condon–Shortley phase $(-1)^m$, which is presented in the lcao orbitals of ABACUS.

7.2 Generating atomic orbital bases

Users may also choose to generate their own atomic obitals. In ABACUS, the atomic orbital bases are generated using a scheme developed in the paper. A detailed description of the procedure for generating orbitals will be provided later.

7.3 BSSE Correction

For treating BSSE(Basis Set Superposition Error), we allow for the inclusion of "empty" or "ghost" atoms in the calculation. Namely, when expanding the Hamiltonian, basis sets on the atoms are used, while the ionic potentials on those atoms are not included when constructing the Hamiltonian.

An empty atom is defined in the `STRU` file when an element name contains the "empty" suffix, such as "H_empty", "O_empty" and so on. Here we provide an example of calculating the molecular formation energy of H_2O with BSSE correction.

In the example, we provide four STRU files:

- STRU_0 : used along with ntype = 2;normal calculation of water molecule ($E(\mathrm{H_2O})$)

  obtained total energy of -466.4838149140513 eV

- STRU_1 : used along with ntype = 2;calculation of single O atom (E_O)

  obtained total energy of -427.9084406198214 eV

- STRU_2 : used along with ntype = 3;calculation of 1st H atom (E_{H1})

  obtained total energy of -12.59853381731160 eV

- STRU_3 : used along with ntype = 3;calculation of 2nd H atom (E_{H2})

  obtained total energy of -12.59853378720844 eV

  Note : Remember to adjust the parameter `ntype` in INPUT file

Thus, the formation energy is given by:

$$\Delta E(\mathrm{H_2O}) = E(\mathrm{H_2O}) - E(\mathrm{O}) - E(\mathrm{H}^1) - E(\mathrm{H}^2) \approx -13.38eV$$

7.4 Pseudopotentials

In ABACUS, we support norm-conserving and ultrasoft pseudopotentials. For norm-conserving pseudopotentials, we support four different formats of the pseudopotential files: UPF, UPF2, VWR, and BLPS. For ultrasoft pseudopotentials, currently we support only one format of the pseudopotential files: UPF2.

For more information, check the `ATOMIC_SPECIES` section in the specification of the *STRU file*.

Here we list some common sources of the pseudopotential files:

1. Quantum ESPRESSO.
2. SG15-ONCV.
3. DOJO.
4. BLPS.

CHAPTER

EIGHT

GEOMETRY OPTIMIZATION

By setting `calculation` to be `relax` or `cell-relax`, ABACUS supports structural relaxation and variable-cell relaxation.

Current implementation of variable-cell relaxation in ABACUS now follows a nested procedure: fixed cell structural relaxation will be performed, followed by an update of the cell parameters, and the process is repeated until convergence is achieved.

An example of the variable cell relaxation can be found in our repository, which is provided with the reference output file log.ref. Note that in log.ref, each ionic step is labelled in the following manner:

```
----------------------------------------
RELAX CELL : 3
RELAX IONS : 1 (in total: 15)
----------------------------------------
```

indicating that this is the first ionic step of the 3rd cell configuration, and it is the 15-th ionic step in total.

8.1 Optimization Algorithms

In the nested procedure mentioned above, we used CG method to perform cell relaxation, while offering four different algorithms for doing fixed-cell structural relaxation: BFGS, SD(steepest descent), CG(conjugate gradient), as well as a mixed CG-BFGS method. The optimziation algorithm can be selected using keyword *relax_method*. We also provide a *list of keywords* for controlling the relaxation process.

8.1.1 BFGS method

The BFGS method is a quasi-Newton method for solving nonlinear optimization problem. It belongs to the class of quasi-Newton method where the Hessian matrix is approximated during the optimization process. If the initial point is not far from the extrema, BFGS tends to work better than gradient-based methods.

In ABACUS, we implemented the BFGS method for doing fixed-cell structural relaxation.

8.1.2 SD method

The SD (steepest descent) method is one of the simplest first-order optimization methods, where in each step the motion is along the direction of the gradient, where the function descents the fastest.

In practice, SD method may take many iterations to converge, and is generally not used.

8.1.3 CG method

The CG (conjugate gradient) method is one of the most widely used methods for solving optimization problems.

In ABACUS, we implemented the CG method for doing fixed-cell structural relaxation as well as the optimization of cell parameters.

8.2 Constrained Optimization

Apart from conventional optimization where all degrees of freedom are allowed to move, we also offer the option of doing constrained optimization in ABACUS.

8.2.1 Fixing Atomic Positions

Users may note that in the above-mentioned example, the atomic positions in STRU file are given along with three integers:

```
Al
0.0
4
0.00 0.00 0.00 1 1 1
0.53 0.50 0.00 1 1 1
0.50 0.00 0.52 1 1 1
0.00 0.50 0.50 1 1 1
```

For relaxation calculations, the three integers denote whether the corresponding degree of freedom is allowed to move. For example, if we replace the STRU file by:

```
Al
0.0
4
0.00 0.00 0.00 1 1 0
0.53 0.50 0.00 1 1 1
0.50 0.00 0.52 1 1 1
0.00 0.50 0.50 1 1 1
```

then the first Al atom will not be allowed to move in z direction.

Fixing atomic position is sometimes helpful during relaxation of isolated molecule/cluster, to prevent the system from drifting in space.

8.2.2 Fixing Cell Parameters

Sometimes we want to do variable-cell relaxation with some of the cell degrees of freedom fixed. This is achieved by keywords such as *fixed_axes*, *fixed_ibrav* and *fixed_atoms*. Specifically, if users are familiar with the `ISIF` option from VASP, then we offer the following correspondence:

- ISIF = 0 : calculation = “relax”
- ISIF = 1, 2 : calculation = “relax”, cal_stress = 1
- ISIF = 3 : calculation = “cell-relax”
- ISIF = 4 : calculation = “cell-relax”, fixed_axes = “volume”
- ISIF = 5 : calculation = “cell-relax”, fixed_axes = “volume”, fixed_atoms = True
- ISIF = 6 : calculation = “cell-relax”, fixed_atoms = True
- ISIF = 7 : calculation = “cell-realx”, fixed_axes = “shape”, fixed_atoms = True

CHAPTER

NINE

MOLECULAR DYNAMICS

Molecular dynamics (MD) is a computer simulation method for analyzing the physical movements of atoms and molecules. The atoms and molecules are allowed to interact for a fixed period of time, giving a view of the dynamic "evolution" of the system. In the most common version, the trajectories of atoms and molecules are determined by numerically solving Newton's equations of motion for a system of interacting particles, where forces between the particles and their potential energies are calculated using first-principles calculations (first-principles molecular dynamics, FPMD), or interatomic potentials and molecular mechanics force fields (classical molecular dynamics, CMD).

By setting *calculation* to be `md`, ABACUS currently provides several different MD evolution methods, which is specified by keyword *md_type* in the `INPUT` file:

- fire: a MD-based relaxation algorithm, see *details* here
- nve: NVE ensemble with velocity Verlet algorithm
- nvt: NVT ensemble
- npt: Nose-Hoover style NPT ensemble
- langevin: NVT ensemble with Langevin thermostat
- msst: MSST method

When *md_type* is set to nvt, *md_thermostat* is used to specify the temperature control method used in NVT ensemble.

- nhc: Nose-Hoover chain
- anderson: Anderson thermostat
- berendsen: Berendsen thermostat
- rescaling: velocity Rescaling method 1
- rescale_v: velocity Rescaling method 2

When *md_type* is set to npt, *md_pmode* is used to specify the cell fluctuation mode in NPT ensemble based on the Nose-Hoover style non-Hamiltonian equations of motion.

- iso: isotropic cell fluctuations
- aniso: anisotropic cell fluctuations
- tri: non-orthogonal (triclinic) simulation box

Furthermore, ABACUS also provides a *list of keywords* to control relevant parmeters used in MD simulations.

The MD output information will be written into the file `MD_dump`[?] in which the atomic forces, atomic velocities, and lattice virial are controlled by keyword *dump_force*, *dump_vel*, and *dump_virial*, respectively.

Examples of MD simulations are also provided. There are eight INPUT files corresponding to eight different MD evolution methods in the directory. For examlpe, `INPUT_0` shows how to employ the NVE simulation.

To run any of the fix cases, users may enter the directory, copy the corresponding input file to `INPUT`, and run ABACUS.

9.1 FIRE

FIRE (fast inertial relaxation engine) is a MD-based minimization algorithm. It is based on conventional molecular dynamics with additional velocity modifications and adaptive time steps. The MD trajectory will descend to an energy-minimum.

9.2 NVE

NVE ensemble (i. e. microcanonical ensemble) is a statistical ensemble that represents the possible states of a mechanical system whose total energy is exactly specified. The system is assumed to be isolated in the sense that it cannot exchange energy or particles with its environment, so that the energy of the system does not change with time.

The primary macroscopic variables of the microcanonical ensemble are the total number of particles in the system (symbol: N), the system's volume (symbol: V), as well as the total energy in the system (symbol: E). Each of these is assumed to be constant in the ensemble.

Currently NVE ensemble in ABACUS is implemented based on the velocity verlet algorithm.

9.3 Nose Hoover Chain

NVT ensemble (i. e. canonical ensemble) is the statistical ensemble that represents the possible states of a mechanical system in thermal equilibrium with a heat bath at a fixed temperature. The system can exchange energy with the heat bath, so that the states of the system will differ in total energy.

The principal thermodynamic variable of the canonical ensemble, determining the probability distribution of states, is the absolute temperature (symbol: T). The ensemble typically also depends on mechanical variables such as the number of particles in the system (symbol: N) and the system's volume (symbol: V), each of which influence the nature of the system's internal states. An ensemble with these three parameters is sometimes called the NVT ensemble.

The isothermal–isobaric ensemble (constant temperature and constant pressure ensemble), also called NPT ensemble, is a statistical mechanical ensemble that maintains the number of particles N, constant temperature T, and constant pressure P. This ensemble plays an important role in chemistry as chemical reactions are usually carried out under constant pressure condition. The NPT ensemble is also useful for measuring the equation of state of model systems whose virial expansion for pressure cannot be evaluated, or systems near first-order phase transitions.

ABACUS perform time integration on Nose-Hoover style non-Hamiltonian equations of motion which are designed to generate positions and velocities sampled from NVT and NPT ensemble.

9.4 Langevin

Langevin thermostat can be used for molecular dynamics equations by assuming that the atoms being simulated are embedded in a sea of much smaller fictional particles. In many instances of solute-solvent systems, the behavior of the solute is desired, and the behavior of the solvent is non-interesting(e.g. proteins, DNA, nanoparticles in solution). In these cases, the solvent influences the dynamics of the solute(typically nanoparticles) via random collisions, and by imposing a frictional drag force on the motion of the nanoparticle in the solvent. The damping factor and the random force combine to give the correct NVT ensemble.

9.5 Anderson

Anderson thermostat couples the system to a heat bath that imposes the desired temperature to simulate the NVT ensemble. The coupling to a heat bath is represented by stochastic collision that act occasionally on randomly selected particles.

9.6 Berendsen

Reset the temperature of a group of atoms by using a Berendsen thermostat, which rescales their velocities every timestep. In this scheme, the system is weakly coupled to a heat bath with some temperature. Though the thermostat does not generate a correct canonical ensemble (especially for small systems), for large systems on the order of hundreds or thousands of atoms/molecules, the approximation yields roughly correct results for most calculated properties.

9.7 Rescaling

Reset the temperature of a group of atoms by explicitly rescaling their velocities. Velocities are rescaled if the current and target temperature differ more than *md_tolerance* (Kelvin).

9.8 Rescale_v

Reset the temperature of a group of atoms by explicitly rescaling their velocities. Every *md_nraise* steps the current temperature is rescaled to target temperature.

9.9 MSST

ABACUS performs the Multi-Scale Shock Technique (MSST) integration to update positions and velocities each timestep to mimic a compressive shock wave passing over the system. The MSST varies the cell volume and temperature in such a way as to restrain the system to the shock Hugoniot and the Rayleigh line. These restraints correspond to the macroscopic conservation laws dictated by a shock front.

9.10 DPMD

Compiling ABACUS with DeePMD-kit, MD calculations based on machine learning DP model is enabled.

To employ DPMD calculations, *esolver_type* should be set to `dp`. And the filename of DP model is specified by keyword *pot_file*.

First, we can find whether contains keyword `type_map` in the DP model through the shell command:

```
strings Al-SCAN.pb | grep type_map
```

```
{"model": {"type_map": ["Al"], "descriptor": {"type": "se_e2_a", "sel": [150], "rcut_
↪smth": 0.5, "rcut": 6.0, "neuron": [25, 50, 100], "resnet_dt": false, "axis_neuron
↪": 16, "seed": 1, "activation_function": "tanh", "type_one_side": false, "precision
↪": "default", "trainable": true, "exclude_types": [], "set_davg_zero": false},
```

(continues on next page)

(continued from previous page)


```
↪"fitting_net": {"neuron": [240, 240, 240], "resnet_dt": true, "seed": 1, "type":
↪"ener", "numb_fparam": 0, "numb_aparam": 0, "activation_function": "tanh",
↪"precision": "default", "trainable": true, "rcond": 0.001, "atom_ener": []}, "data_
↪stat_nbatch": 10, "data_stat_protect": 0.01}, "learning_rate": {"type": "exp",
↪"decay_steps": 5000, "start_lr": 0.001, "stop_lr": 3.51e-08, "scale_by_worker":
↪"linear"}, "loss": {"type": "ener", "start_pref_e": 0.02, "limit_pref_e": 1, "start_
↪pref_f": 1000, "limit_pref_f": 1, "start_pref_v": 0, "limit_pref_v": 0, "start_pref_
↪ae": 0.0, "limit_pref_ae": 0.0, "start_pref_pf": 0.0, "limit_pref_pf": 0.0, "enable_
↪atom_ener_coeff": false}, "training": {"training_data": {"systems": ["../deepmd_
↪data/"], "batch_size": "auto", "set_prefix": "set", "auto_prob": "prob_sys_size",
↪"sys_probs": null}, "validation_data": {"systems": ["../deepmd_validation"], "batch_
↪size": 1, "numb_btch": 3, "set_prefix": "set", "auto_prob": "prob_sys_size", "sys_
↪probs": null}, "numb_steps": 1000000, "seed": 10, "disp_file": "lcurve.out", "disp_
↪freq": 100, "save_freq": 1000, "save_ckpt": "model.ckpt", "disp_training": true,
↪"time_training": true, "profiling": false, "profiling_file": "timeline.json",
↪"enable_profiler": false, "tensorboard": false, "tensorboard_log_dir": "log",
↪"tensorboard_freq": 1}}
```


If the keyword `type_map` is found, ABACUS will match the atom types between `STRU` and DP model.

Otherwise, all atom types must be specified in the `STRU` in the order consistent with that of the DP model, even if the number of atoms is zero!

For example, there is a Al-Cu-Mg ternary-alloy DP model, but the simulated cell is a Al-Cu binary alloy. Then the `STRU` should be written as follows:

```
ATOMIC_SPECIES
Al  26.982
Cu  63.546
Mg  24.305

LATTICE_CONSTANT
1.889727000000

LATTICE_VECTORS
4.0  0.0  0.0
0.0  4.0  0.0
0.0  0.0  4.0

ATOMIC_POSITIONS
Cartesian

Al
0
2
0.0  0.0  0.0
0.5  0.5  0.0

Cu
0
2
0.5  0.0  0.5
0.0  0.5  0.5

Mg
0
0
```

CHAPTER

TEN

ACCELERATE PERFORMANCE

This section describes various methods for improving ABACUS performance for different classes of problems running on different kinds of devices.

Accelerated versions of CUDA GPU implementations have been added to ABACUS, which will typically run faster than the standard non-accelerated versions. This requires appropriate hardware to be present on your system, e.g. NVIDIA GPUs.

10.1 CUDA GPU Implementations

In ABACUS, we provide the option to use the GPU devices to accelerate the performance. And it has the following general features:

- **Full gpu implementations**: During the SCF progress, `Psi`, `Hamilt`, `Hsolver`, `DiagCG`, and `DiagoDavid` classes are stored or calculated by the GPU devices.
- **Electronic state data**: (e.g. electronic density) are moved from the GPU to the CPU(s) every scf step.
- **Acclerated by the NVIDIA libraries**: `cuBLAS` for common linear algebra calculations, `cuSolver` for eigen values/vectors, and `cuFFT` for the conversions between the real and recip spaces.
- **Multi GPU supprted**: Using multiple MPI tasks will often give the best performance. Note each MPI task will be bind to a GPU device with automatically computing load balancing.
- **Parallel strategy**: K point parallel.

10.1.1 Required hardware/software

To compile and use ABACUS in CUDA mode, you currently need to have an NVIDIA GPU and install the corresponding NVIDIA CUDA toolkit software on your system (this is only tested on Linux and unsupported on Windows):

- Check if you have an NVIDIA GPU: cat /proc/driver/nvidia/gpus/*/information
- Go to https://developer.nvidia.com/cuda-downloads
- Install a driver and toolkit appropriate for your system (SDK is not necessary)

10.1.2 Building ABACUS with the GPU support:

Check the Advanced Installation Options for the installation of CUDA version support.

10.1.3 Run with the GPU support by editing the INPUT script:

In `INPUT` file we need to set the value keyword *device* to be `gpu`.

10.1.4 Examples

We provides examples of gpu calculations.

10.1.5 Known limitations

- CG, BPCG and Davidson methods are supported, so the input keyword `ks_solver` can take the values `cg`, `bpcg` or `dav`,
- Only PW basis is supported, so the input keyword `basis_type` can only take the value `pw`,
- Only k point parallelization is supported, so the input keyword `kpar` will be set to match the number of MPI tasks automatically.
- By default, CUDA architectures 60, 70, 75, 80, 86, and 89 are compiled (if supported). It can be overriden using the CMake variable `CMAKE_CUDA_ARCHITECTURES` or the environmental variable `CUDAARCHS`.

10.1.6 FAQ

```
Q: Does the GPU implementations support atomic orbital basis sets?
A: Currently no.
```

CHAPTER

ELEVEN

ELECTRONIC PROPERTIES AND OUTPUTS

11.1 Extracting Band Structure

ABACUS can calculate the energy band structure, and the examples can be found in examples/band. Similar to the DOS case, we first, do a ground-state energy calculation ***with one additional keyword "out_chg" in the INPUT file***:

```
out_chg             1
```

This will produce the converged charge density, which is contained in the file SPIN1_CHG.cube. Then, use the same `STRU` file, pseudopotential file and atomic orbital file (and the local density matrix file onsite.dm if DFT+U is used) to do a non-self-consistent calculation. In this example, the potential is constructed from the ground-state charge density from the proceeding calculation. Now the INPUT file is like:

```
INPUT_PARAMETERS
#Parameters (General)
ntype 1
nbands 8
calculation nscf
basis_type lcao
read_file_dir   ./

#Parameters (Accuracy)
ecutwfc 60
scf_nmax 50
scf_thr 1.0e-9
pw_diag_thr 1.0e-7

#Parameters (File)
init_chg file
out_band 1
out_proj_band 1

#Parameters (Smearing)
smearing_method gaussian
smearing_sigma 0.02
```

Here the the relevant k-point file KPT looks like,

```
K_POINTS # keyword for start
6 # number of high symmetry lines
Line # line-mode
0.5 0.0 0.5 20 # X
0.0 0.0 0.0 20 # G
```

(continues on next page)

(continued from previous page)

```
0.5 0.5 0.5 20 # L
0.5 0.25 0.75 20 # W
0.375 0.375 0.75 20 # K
0.0 0.0 0.0 1 # G
```

This means we are using:

- 6 number of k points, here means 6 k points: (0.5, 0.0, 0.5) (0.0, 0.0, 0.0) (0.5, 0.5, 0.5) (0.5, 0.25, 0.75) (0.375, 0.375, 0.75) (0.0, 0.0, 0.0)
- 20/1 number of k points along the segment line, which is constructed by two adjacent k points.

Run the program, and you will see a file named BANDS_1.dat in the output directory. Plot it to get energy band structure.

If "out_proj_band" set 1, it will also produce the projected band structure in a file called PBAND_1 in xml format.

The PBAND_1 file starts with number of atomic orbitals in the system, the text contents of element `<band structure>` is the same as data in the BANDS_1.dat file, such as:

```
<pband>
<nspin>1</nspin>
<norbitals>153</norbitals>
<band_structure nkpoints="96" nbands="50" units="eV">
...
```

The rest of the files arranged in sections, each section with a header such as below:

```
<orbital
 index="                                    1"
 atom_index="                                    1"
 species="Si"
 l="                                  0"
 m="                                  0"
 z="                                  1"
>
<data>
...
</data>
```

The shape of text contents of element `<data>` is (Number of k-points, Number of bands)

11.2 Calculating DOS and PDOS

11.2.1 DOS

ABACUS can calculate the density of states (DOS) of the system, and the examples can be found in examples/dos. We first, do a ground-state energy calculation ***with one additional keyword "out_chg" in the INPUT file***:

```
out_chg              1
```

this will produce the converged charge density, which is contained in the file SPIN1_CHG.cube. Then, use the same `STRU` file, pseudopotential file and atomic orbital file (and the local density matrix file onsite.dm if DFT+U is used) to do a non-self-consistent calculation. In this example, the potential is constructed from the ground-state charge density from the proceeding calculation. Now the INPUT file is like:

```
INPUT_PARAMETERS
#Parameters (General)
suffix Si2_diamond
ntype 1
nbands 8
calculation nscf
basis_type lcao
read_file_dir   ./

#Parameters (Accuracy)
ecutwfc 60
symmetry 1
scf_nmax 50
scf_thr 1.0e-9
pw_diag_thr 1.0e-7

#Parameters (File)
init_chg file
out_dos 1
dos_sigma 0.07
```

Some parameters in the INPUT file are explained:

- calculation

  choose which kind of calculation: scf calculation, nscf calculation, structure relaxation or Molecular Dynamics. Now we need to do one step of nscf calculation. Attention: This is a main variable of ABACUS, and for its more information please see the here.

- pw_diag_thr

  threshold for the CG method which diagonalizes the Hamiltonian to get eigenvalues and eigen wave functions. If one wants to do nscf calculation, pw_diag_thr needs to be changed to a smaller account, typically smaller than 1.0e-3. Note that this parameter only apply to plane-wave calculations that employ the CG or Davidson method to diagonalize the Hamiltonian. For its more information please see the here.

  For LCAO calculations, this parameter will be neglected !

- init_chg

  the type of starting density. When doing scf calculation, this variable can be set "atomic". When doing nscf calculation, the charge density already exists(eg. in SPIN1_CHG.cube), and the variable should be set as "file". It means the density will be read from the existing file SPIN1_CHG.cube. For its more information please see the here.

- out_dos

  output density of state(DOS). The unit of DOS is `(number of states)/(eV * unitcell)`. For its more information please see the here.

- dos_sigma

  the gaussian smearing parameter(DOS), in unit of eV. For its more information please see the here.

- read_file_dir

  the location of electron density file. For its more information please see the here.

To have an accurate DOS, one needs to have a denser k-point mesh. For example, the KPT file can be set as:

```
K_POINTS
0
Gamma
8 8 8 0 0 0
```

Run the program, and you will see a file named DOS1_smearing.dat in the output directory. The first two columns in the file are the energy and DOS, respectively, and the third column is the sum of DOS. Plot file DOS1_smearing.dat with graphing software, and you'll get the DOS.

```
            -5.49311           0.0518133           0.0518133
            -5.48311           0.0641955            0.116009
            -5.47311           0.0779299            0.193939
            -5.46311           0.0926918             0.28663
            -5.45311            0.108023            0.394653
            -5.44311            0.123346            0.517999
            ...
```

11.2.2 PDOS

Along with the DOS1_smearing.dat file, we also produce the projected density of states (PDOS) in a file called PDOS.

The PDOS file starts with number of atomic orbitals in the system, then a list of energy values, such as:

```
<pdos>
<nspin>1</nspin>
<norbitals>26</norbitals>
<energy_values units="eV">
            -5.50311
            -5.49311
            -5.48311
            -5.47311
...
```

The rest of the fileis arranged in sections, each section with a header such as below:

```
<orbital
 index="                                    1"
 atom_index="                                    1"
 species="Si"
 l="                                 0"
 m="                                 0"
 z="                                 1"
>
<data>
...
</data>
```

which tells the atom and symmetry of the current atomic orbital, and followed by the PDOS values. The values can thus be plotted against the energies. The unit of PDOS is also `(number of states)/(eV * unitcell)`.

11.3 Mulliken Charge Analysis

From version 2.1.0, ABACUS has the function of Mulliken population analysis. The example can be found in examples/mulliken.
To use this function, set *out_mul* to `1` in the INPUT file. After calculation, there will be an output file named `mulliken.txt` in the output directory. In MD calculations, the output interval is controlled by the keyword *out_interval*. In the file, there are contents like (`nspin 1`):

```
STEP: 0
CALCULATE THE MULLIkEN ANALYSIS FOR EACH ATOM
 Total charge of spin 1:        8
 Total charge:        8
Decomposed Mulliken populations
0                  Zeta of Si                        Spin 1
s                         0                       1.2553358
  sum over m                                      1.2553358
s                         1                     -0.030782972
  sum over m                                    -0.030782972
  sum over m+zeta                                 1.2245529
pz                         0                      0.85945806
px                         0                      0.85945806
py                         0                      0.85945806
  sum over m                                      2.5783742
pz                         1                     0.0065801228
px                         1                     0.0065801228
py                         1                     0.0065801228
  sum over m                                     0.019740368
  sum over m+zeta                                 2.5981145
dz^2                         0                       0.0189287
dxz                          0                      0.046491729
dyz                          0                      0.046491729
dx^2-y^2                           0                       0.0189287
dxy                          0                      0.046491729
  sum over m                                     0.17733259
  sum over m+zeta                                0.17733259
Total Charge on atom:  Si                   4
 ...
```

The file gives Mulliken charge in turn according to the order of atoms in the system. For example, the following block is for the first atom in system (`nspin 2`),

```
0            Zeta of Si               Spin 1              Spin 2                Sum  ␣
↪              Diff
...
Total Charge on atom:  Si                    4
Total Magnetism on atom:  Si      -1.2739809e-14
```

And the next block is for the second atom in system, and so on.

```
1            Zeta of Si               Spin 1              Spin 2                Sum  ␣
↪              Diff
...
```

For each atom, the file gives detailed Mulliken population analysis at different levels,

- magnetic quantum number level: such as lines beigin with 's,px,py,pz,...'
- azimuthal quantum number level: such as lines begin with 'sum over m'.

- principal quantum number level: such as lines begin with 'sum over m+zeta'. Here 'zeta' equals 'zeta' in the file, which means how many radial atomic orbitals there are for a given orbital angular momentum.
- atomic level: such as lines begin with 'Total Charge on atom'.

More orbital information can be found in 'Orbital' file output with 'mulliken.txt' when `out_mul 1`

11.4 Extracting Electrostatic Potential

From version 2.1.0, ABACUS has the function of outputing electrostatic potential, which consists of Hartree potential and the local pseudopotential. To use this function, set 'out_pot' to '2' in the INPUT file. Here is an example for the Si-111 surface, and the INPUT file is:

```
INPUT_PARAMETERS
#Parameters (1.General)
calculation scf
ntype 1
nbands 100
gamma_only 0

#Parameters (2.Iteration)
ecutwfc 50
scf_thr 1e-8
scf_nmax 200

#Parameters (3.Basis)
basis_type lcao
ks_solver genelpa

#Parameters (4.Smearing)
smearing_method gaussian
smearing_sigma 0.01

#Parameters (5.Mixing)
mixing_type broyden
mixing_beta 0.4
out_pot 2
```

The STRU file is:

```
ATOMIC_SPECIES
Si 1.000 Si_ONCV_PBE-1.0.upf

NUMERICAL_ORBITAL
Si_gga_8au_60Ry_2s2p1d.orb

LATTICE_CONSTANT
1.8897162

LATTICE_VECTORS
7.6800298691 0.0000000000 0.0000000000
-3.8400149345 6.6511009684 0.0000000000
0.0000000000 0.0000000000 65.6767997742

ATOMIC_POSITIONS
Cartesian
```

(continues on next page)

(continued from previous page)

```
Si
0.0
40
3.840018749 2.217031479 2.351520061 0 0 0
3.840014935 0.000000000 3.135360003 0 0 0
3.840018749 2.217031479 5.486879826 0 0 0
3.840014935 0.000000000 6.270720005 0 0 0
3.840018749 2.217031479 8.622240067 0 0 0
3.840014935 0.000000000 9.406080246 0 0 0
3.840018749 2.217031479 11.757599831 0 0 0
3.840014935 0.000000000 12.541440010 0 0 0
3.840018749 2.217031479 14.892959595 0 0 0
3.840014935 0.000000000 0.000000000 0 0 0
1.920011044 5.542582035 2.351520061 0 0 0
1.920007467 3.325550556 3.135360003 0 0 0
1.920011044 5.542582035 5.486879826 0 0 0
1.920007467 3.325550556 6.270720005 0 0 0
1.920011044 5.542582035 8.622240067 0 0 0
1.920007467 3.325550556 9.406080246 0 0 0
1.920011044 5.542582035 11.757599831 0 0 0
1.920007467 3.325550556 12.541440010 0 0 0
1.920011044 5.542582035 14.892959595 0 0 0
1.920007467 3.325550556 0.000000000 0 0 0
0.000003815 2.217031479 2.351520061 0 0 0
0.000000000 0.000000000 3.135360003 0 0 0
0.000003815 2.217031479 5.486879826 0 0 0
0.000000000 0.000000000 6.270720005 0 0 0
0.000003815 2.217031479 8.622240067 0 0 0
0.000000000 0.000000000 9.406080246 0 0 0
0.000003815 2.217031479 11.757599831 0 0 0
0.000000000 0.000000000 12.541440010 0 0 0
0.000003815 2.217031479 14.892959595 0 0 0
0.000000000 0.000000000 0.000000000 0 0 0
-1.920003772 5.542582035 2.351520061 0 0 0
-1.920007467 3.325550556 3.135360003 0 0 0
-1.920003772 5.542582035 5.486879826 0 0 0
-1.920007467 3.325550556 6.270720005 0 0 0
-1.920003772 5.542582035 8.622240067 0 0 0
-1.920007467 3.325550556 9.406080246 0 0 0
-1.920003772 5.542582035 11.757599831 0 0 0
-1.920007467 3.325550556 12.541440010 0 0 0
-1.920003772 5.542582035 14.892959595 0 0 0
-1.920007467 3.325550556 0.000000000 0 0 0
```

the KPT file is:

```
K_POINTS
0
Gamma
4 4 2 0 0 0
```

Run the program, and you will see the following two files in the output directory,

- ElecStaticPot.cube: contains electrostatic potential (unit: Rydberg) in realspace. This file can be visually viewed by the software of VESTA.

11.5 Extracting Wave Functions

ABACUS is able to output electron wave functions in both PW and LCAO basis calculations. One can find the examples in examples/wfc.

11.5.1 wave function in G space

For the wave function in G space, one only needs to do a ground-state energy calculation with one additional keyword in the INPUT file: '***out_wfc_pw***' for PW basis calculation, and '***out_wfc_lcao***' for LCAO basis calculation. In the PW basis case, the wave function is output in a file called `WAVEFUNC${k}.txt`, where `${k}` is the index of K point.
In the LCAO basis case, several `LOWF_K_${k}.dat` files will be output in multi-k calculation and `LOWF_GAMMA_S1.dat` in gamma-only calculation.

11.5.2 wave function in real space

One can also choose to output real-space wave function in PW basis calculation with the key word ***out_wfc_r***.

After calculation, an additional directory named `wfc_realspace` will appear in the `OUT.${system}` directory.

Notice: when the ***basis_type*** is `lcao`, only `get_wf` ***calculation*** is effective. An example is examples/wfc/lcao_ienvelope_Si2.

11.6 Extracting Charge Density

ABACUS can output the charge density by adding the keyword out_chg in INPUT file:

```
out_chg             1
```

After finishing the calculation, the information of the charge density is stroed in files `OUT.${suffix}/SPIN${spin}_CHG.cube`, which can be used to do visualization. The SPIN${spin}_CHG.cube file looks like:

```
Cubefile created from ABACUS SCF calculation
2 (nspin) 0.914047 (fermi energy, in Ry)
2 0.0 0.0 0.0
27 0.222222 0 0
27 0 0.222222 0
27 0 0 0.222222
 26 16 0 0 0
 26 16 3 3 3
 6.63594288898e-01 8.42344790519e-01 1.16349621677e+00 1.18407505276e+00 8.
↪04461725175e-01 3.77164277045e-01
 1.43308127341e-01 5.93894932356e-02 3.23036576611e-02 2.08414809212e-02 1.
↪51271068218e-02 1.27012859512e-02
 1.15620162933e-02 1.08593210023e-02 1.08593210023e-02 1.15620162933e-02 1.
↪27012859512e-02 1.51271068218e-02
 2.08414809212e-02 3.23036576611e-02 5.93894932356e-02 1.43308127341e-01 3.
↪77164277045e-01 8.04461725175e-01
 1.18407505276e+00 1.16349621677e+00 8.42344790519e-01
 8.42344790519e-01 9.86194056340e-01 1.21545550606e+00 1.14987597026e+00 7.
↪50033272229e-01 3.46047149862e-01
 1.32713411550e-01 5.65432381171e-02 3.13971442033e-02 2.04281058891e-02 1.
↪49536046293e-02 1.26489807288e-02
```

(continues on next page)

(continued from previous page)

```
 1.15432695307e-02 1.08422207044e-02 1.08422207044e-02 1.15432695307e-02 1.
↪26489807288e-02 1.49536046293e-02
 2.04281058891e-02 3.13971442033e-02 5.65432381171e-02 1.32713411550e-01 3.
↪46047149862e-01 7.50033272229e-01
 1.14987597026e+00 1.21545550606e+00 9.86194056340e-01
 ...
```

The first line is a brief description.
The second line contains NSPIN and Fermi energy.
The following 4 lines are the informations of lattice, in order:
  total number of atoms, the coordinate of original point.
  the number of lattice points along lattice vector a1 (nx), a1/nx, in Bohr.
  the number of lattice points along lattice vector a2 (ny), a2/ny, in Bohr.
  the number of lattice points along lattice vector a3 (nz), a3/nz, in Bohr.
The following lines are about the elements and coordinates, in order: the atom number of each atoms, the electron number in pseudopotential, the Cartesian coordinates, in Bohr.
The rest lines are the value of charge density at each grid. Note that the inner loop is z index, followed by y index, x index in turn.
The examples can be found in examples/charge_density

11.7 Extracting Hamiltonian and Overlap Matrices

In ABACUS, we provide the option to write the Hamiltonian and Overlap matrices to files after SCF calculation.

For periodic systems, there are two ways to represent the matrices, the first is to write the entire square matrices for each k point, namely $H(k)$ and $S(k)$; the second is the R space representation, $H(R)$ and $S(R)$, where R is the lattice vector. The two representations are connected by Fourier transform:

- $H(k) = \sum_R H(R)e^{-ikR}$

and

- $S(k) = \sum_R S(R)e^{-ikR}$

11.7.1 out_mat_hs

Users may set the keyword *out_mat_hs* to true for outputting the upper triangular part of the Hamiltonian matrices and overlap matrices for each k point into files in the directory `OUT.${suffix}`. It is available for both gamma_only and multi-k calculations.

The files are named `data-$k-H` and `data-$k-S`, where `$k` is a composite index consisting of the k point index as well as the spin index. The corresponding sequence of the orbitals can be seen in *Basis Set*.

For nspin = 1 and nspin = 4 calculations, there will be only one spin component, so `$k` runs from 0 up to `$nkpoints-1`. For nspin = 2, `$k` runs from `2*$nkpoints-1`. In the latter case, the files are arranged into blocks of up and down spins. For example, if there are 3 k points, then we have the following correspondence:

- data-0-H : 1st k point, spin up
- data-1-H : 2nd k point, spin up
- data-2-H : 3rd k point, spin up
- data-3-H : 1st k point, spin down
- data-4-H : 2nd k point, spin down

- data-5-H : 3rd k point, spin down

As for information on the k points, one may look for the `SETUP K-POINTS` section in the running log.

The first number of the first line in each file gives the size of the matrix, namely, the number of atomic basis functions in the system.

The rest of the file contains the upper triangular part of the specified matrices. For multi-k calculations, the matrices are Hermitian and the matrix elements are complex; for gamma-only calculations, the matrices are symmetric and the matrix elements are real.

11.7.2 out_mat_hs2

The output of R-space matrices is controlled by the keyword *out_mat_hs2*. This functionality is not available for gamma_only calculations. To generate such matrices for gamma only calculations, users should turn off *gamma_only*, and explicitly specify that gamma point is the only k point in the KPT file.

For single-point SCF calculations, if nspin = 1 or nspin = 4, two files `data-HR-sparse_SPIN0.csr` and `data-SR-sparse_SPIN0.csr` are generated, which contain the Hamiltonian matrix $H(R)$ and overlap matrix $S(R)$ respectively. For nspin = 2, three files `data-HR-sparse_SPIN0.csr` and `data-HR-sparse_SPIN1.csr` and `data-SR-sparse_SPIN0.csr` are created, where the first two contain $H(R)$ for spin up and spin down, respectively.

As for molecular dynamics calculations, the format is controlled by *out_interval* and *out_app_flag* in the same manner as the position matrix as detailed in *out_mat_r*.

Each file or each section of the appended file starts with three lines, the first gives the current ion/md step, the second gives the dimension of the matrix, and the last indicates how many different `R` are in the file.

The rest of the files are arranged in blocks. Each block starts with a line giving the lattice vector `R` and the number of nonzero matrix elements, such as:

```
-3 1 1 1020
```

which means there are 1020 nonzero elements in the (-3,1,1) cell.

If there is no nonzero matrix element, then the next block starts immediately on the next line. Otherwise, there will be 3 extra lines in the block, which gives the matrix in CSR format. According to Wikipedia:

The CSR format stores a sparse m × n matrix M in row form using three (one-dimensional) arrays (V, COL_INDEX, ROW_INDEX). Let NNZ denote the number of nonzero entries in M. (Note that zero-based indices shall be used here.)

- The arrays V and COL_INDEX are of length NNZ, and contain the non-zero values and the column indices of those values respectively.
- The array ROW_INDEX is of length m + 1 and encodes the index in V and COL_INDEX where the given row starts. This is equivalent to ROW_INDEX[j] encoding the total number of nonzeros above row j. The last element is NNZ , i.e., the fictitious index in V immediately after the last valid index NNZ - 1.

11.7.3 get_S

We also offer the option of only calculating the overlap matrix without running SCF. For that purpose, in `INPUT` file we need to set the value keyword *calculation* to be `get_S`.

A file named `SR.csr` will be generated in the working directory, which contains the overlap matrix.

11.7.4 examples

We provide examples of outputting the matrices. There are four examples:

- out_hs2_multik : writing H® and S® for multi-k calculation
- out_hs_gammaonly : writing H(k) and S(k) for gamma-only calculation
- out_hs_multik : writing H(k) and S(k) for multi-k calculation
- out_s_multik : running get_S for multi-k calculation

Reference output files are provided in each directory.

11.8 Extracting Density Matrices

ABACUS can output the density matrix by adding the keyword "out_dm" in INPUT file:

```
out_dm              1
```

After finishing the calculation, the information of the density matrix is stroed in files `OUT.${suffix}/SPIN${spin}_DM`, which looks like:

```
test
 5.39761
 0.5 0.5 0
 0.5 0 0.5
 0 0.5 0.5
 Si
 2
Direct
 0 0 0
 0.25 0.25 0.25

 1
 0.570336288801065 (fermi energy)
  26 26

 3.904e-01 1.114e-02 2.050e-14 1.655e-13 1.517e-13 -7.492e-15 -1.729e-14 5.915e-15
 -9.099e-15 2.744e-14 3.146e-14 6.631e-15 2.594e-15 3.904e-01 1.114e-02 -7.395e-15
 ...
```

The first 5 lines are the informations of lattice, in order:
lattice name (if keyword latname is not specified in INPUT, this will be "test"),
lattice constance with unit in angstrom,
lattice vector a,
lattice vector b,
lattice vector c.
The following lines are about the elements and coordinates, in order: all elements, the atom number of each elements, the

type of coordinate, the coordinates.
After a blank line, the output is the values of NSPIN and fermi energy.
The following line is dimension of the density matrix, and the rest lines are the value of each matrix element.

The examples can be found in examples/density_matrix

- Note: now this function is valid only for LCAO gamma only calcualtion.

11.9 Berry Phase Calculation

From version 2.0.0, ABACUS is capable of calculating macroscopic polarization of insulators by using the Berry phase method, known as the "modern theory of polarization". To calculate the polarization, you need first to do a self-consistent calculation to get the converged charge density. Then, do a non-self-consistent calculation with berry_phase setting to 1. You need also to specify the direction of the polarization you want to calculate. An example is given in the directory examples/berryphase/lcao_PbTiO3.

To run this example, first do a self-consistent calculation:

```
cp INPUT-scf INPUT
cp KPT-scf KPT
mpirun -np 4 abacus
```

Then run a non-self-consistent berry-phase calculation:

```
cp INPUT-nscf-c INPUT
cp KPT-nscf-c KPT
mpirun -np 4 abacus
```

In this example, we calculate the electric polarization along c axis for PbTiO~3~, and below are the INPUT file (nscf) and KPT file (nscf):

```
INPUT_PARAMETERS
pseudo_dir      ../../../tests/PP_ORB  //the path to locate the pesudopotential files
orbital_dir     ../../../tests/PP_ORB  //the path to locate the numerical orbital␣
↪files
ntype           3
ecutwfc         50 // Ry
symmetry        0 // turn off symmetry
calculation     nscf // non-self-consistent calculation
basis_type      lcao // atomic basis
init_chg  file // read charge from files
berry_phase     1 // calculate Berry phase
gdir            3 // calculate polarization along c axis
```

Note: You need to turn off the symmetry when do Berry phase calculations. Currently, ABACUS support Berry phase calculation with nspin=1 and nspin=2. The Berry phase can be calculated in both pw and lcao bases.

- *berry_phase* : 1, calculate berry phase; 0, no calculate berry phase.
- *gdir* : 1, 2, 3, the lattice vector direction of the polarization you want to calculate.

The KPT file need to be modified according to gdir in the INPUT file. Generally, you need denser k points along this direction. For example, in the following KPT file, 4 k-points are taken along the a and b axes, and 8 k-points are taken along the c-axis. You should check the convergence of the k points when calculating the polarization.

```
K_POINTS
0
```

(continues on next page)

(continued from previous page)

```
Gamma
4 4 8 0 0 0
```

The results of the berry phase calculation are written in the "running_nscf.log" in the OUT folder. You may search for these results by searching for keywords "POLARIZATION CALCULATION".

The results are shown as follows:

```
 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 |                                                                              |
 | POLARIZATION CALCULATION:                                                    |
 |                  Modern Theory of Polarization                               |
 | calculate the Macroscopic polarization of a crystalline insulator            |
 | by using Berry Phase method.                                                 |
 |                                                                              |
 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

 VALUES OF POLARIZATION

  The Ionic Phase:   -0.10600
 Electronic Phase:    0.92508

 The calculated polarization direction is in R3 direction

 P =    7.4095194  (mod   18.0922373)  (   0.0000000,   0.0000000,   7.4095194) (e/
↪Omega).bohr

 P =    0.0155792  (mod    0.0380407)  (   0.0000000,   0.0000000,   0.0155792) e/
↪bohr^2

 P =    0.8906925  (mod    2.1748536)  (   0.0000000,   0.0000000,   0.8906925) C/m^2
```

The electric polarization **P** is multivalued, which modulo a quantum e**R**/V~cell~. Note: the values in parentheses are the components of the **P** along the c axis in the x, y, z Cartesian coordinates when set gdir = 3 in INPUT file.

CHAPTER

TWELVE

INTERFACES TO OTHER SOFTWARES

12.1 DeePKS

DeePKS is a machine-learning aided density funcitonal model that fits the energy difference between highly accurate but computationally demanding method and effcient but less accurate method via neural-network. As such, the trained DeePKS model can provide highly accurate energetics (and forces) with relatively low computational cost, and can therefore act as a bridge to connect expensive quantum mechanic data and machine-learning-based potentials. While the original framework of DeePKS is for molecular systems, please refer to this reference for the application of DeePKS in periodic systems.

Detailed instructions on installing and running DeePKS can be found on this website. An example for training DeePKS model with ABACUS is also provided. The DeePKS-related keywords in `INPUT` file can be found here.

Note: Use the LCAO basis for DeePKS-related calculations

12.2 DP-GEN

DP-GEN, the deep potential generator, is a package designed to generate deep learning based model of interatomic potential energy and force fields (Yuzhi Zhang, Haidi Wang, Weijie Chen, Jinzhe Zeng, Linfeng Zhang, Han Wang, and Weinan E, DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models, Computer Physics Communications, 2020, 107206). ABACUS can now interface with DP-GEN to generate deep potentials and perform autotests. The minimum recommended version is ABACUS 3.0, dpdata 0.2.8, and dpgen 0.10.7 . In the following part, we take the FCC aluminum as an example.

12.2.1 init_bulk and run

This example can be found in examples/dpgen-example/init_and_run directory.

Firstly, one needs to prepare input files for ABACUS calculation, e.g., "INPUT", "INPUT.md", "KPT", "Al.STRU", "Al_ONCV_PBE-1.0.upf", which are the main input file containing input tags, k-point mesh, crystal structure and pseudoptential, respectively. "INPUT" is for scf calculation, and "INPUT.md" is for AIMD (ab-initio molecular dynamic) calculation.

Secondly, for the "dpgen init_bulk" step, an `init.json` file should be provided:

```
{
    "init_fp_style":     "ABACUS",    # abacus interface
    "stages":            [1,2,3,4],
    "cell_type":         "fcc",
    "super_cell":        [2, 1, 1],
```

(continues on next page)

(continued from previous page)

```
    "elements":          ["Al"],
    "from_poscar":       true,
    "from_poscar_path": "./Al.STRU",
    "potcars":           ["Al_ONCV_PBE-1.0.upf"],
    "relax_incar":       "INPUT",
    "relax_kpt":         "KPT",
    "md_incar" :         "INPUT.md",
    "md_kpt" :           "KPT",
    "skip_relax":        false,
    "scale":             [1.00],
    "pert_numb":         10,
    "pert_box":          0.01,
    "pert_atom":         0.01,
    "coll_ndata":        10,
    "_comment":          "that's all"
}
```

Next, for the "dpgen run" step, the following `run_param.json` should be provided.

```
{
    "type_map": [
        "Al"
    ],
    "mass_map": [
        26.9815
    ],
    "init_data_prefix": "./",
    "init_data_sys": [
        "Al.STRU.01x01x01/02.md/sys-0004/deepmd"
    ],
    "sys_format": "abacus/stru",    # the initial structures are in ABACUS/STRU formate
    "sys_configs_prefix": "./",
    "sys_configs": [
        [
            "Al.STRU.01x01x01/01.scale_pert/sys-0004/scale*/00000*/STRU"
        ],
        [
            "Al.STRU.01x01x01/01.scale_pert/sys-0004/scale*/000010/STRU"
        ]
    ],
    "_comment": " 00.train ",
    "numb_models": 4,
    "default_training_param": {
        "model": {
            "type_map": [
                "Al"
            ],
            "descriptor": {
                "type": "se_e2_a",
                "sel": [
                    16
                ],
                "rcut_smth": 0.5,
                "rcut": 5.0,
                "neuron": [
                    10,
                    20,
```

(continues on next page)

(continued from previous page)

```
                    40
                ],
                "resnet_dt": true,
                "axis_neuron": 12,
                "seed": 1
            },
            "fitting_net": {
                "neuron": [
                    25,
                    50,
                    100
                ],
                "resnet_dt": false,
                "seed": 1
            }
        },
        "learning_rate": {
            "type": "exp",
            "start_lr": 0.001,
            "decay_steps": 100
        },
        "loss": {
            "start_pref_e": 0.02,
            "limit_pref_e": 2,
            "start_pref_f": 1000,
            "limit_pref_f": 1,
            "start_pref_v": 0.0,
            "limit_pref_v": 0.0
        },
        "training": {
            "stop_batch": 20000,
            "disp_file": "lcurve.out",
            "disp_freq": 1000,
            "numb_test": 4,
            "save_freq": 1000,
            "save_ckpt": "model.ckpt",
            "disp_training": true,
            "time_training": true,
            "profiling": false,
            "profiling_file": "timeline.json",
            "_comment": "that's all"
        }
    },
    "_comment": "01.model_devi ",
    "model_devi_dt": 0.002,
    "model_devi_skip": 0,
    "model_devi_f_trust_lo": 0.05,
    "model_devi_f_trust_hi": 0.15,
    "model_devi_clean_traj": false,
    "model_devi_jobs": [
        {
            "sys_idx": [
                0
            ],
            "temps": [
                50
            ],
```

(continues on next page)

(continued from previous page)

```
            "press": [
                1.0
            ],
            "trj_freq": 10,
            "nsteps": 300,
            "ensemble": "nvt",
            "_idx": "00"
        },
        {
            "sys_idx": [
                1
            ],
            "temps": [
                50
            ],
            "press": [
                1.0
            ],
            "trj_freq": 10,
            "nsteps": 3000,
            "ensemble": "nvt",
            "_idx": "01"
        }
    ],
    "fp_style": "abacus/scf",
    "shuffle_poscar": false,
    "fp_task_max": 20,
    "fp_task_min": 5,
    "fp_pp_path": "./",
    "fp_pp_files": ["Al_ONCV_PBE-1.0.upf"],    # the pseudopotential file
    "fp_orb_files": ["Al_gga_9au_100Ry_4s4p1d.orb"],  # the orbital file (use only in␣
↪LCAO calculation)
    "k_points":[2, 2, 2, 0, 0, 0],  # k-mesh setting
    "user_fp_params":{  # All the ABACUS input paramters are defined here
    "ntype": 1,          # defining input parameters from INPUT files is not supported␣
↪yet.
    "ecutwfc": 80,
    "mixing_type": "broyden",
    "mixing_beta": 0.8,
    "symmetry": 1,
    "nspin": 1,
    "ks_solver": "cg",
    "smearing_method": "mp",
    "smearing_sigma": 0.002,
    "scf_thr":1e-8,
    "cal_force":1,          # calculate force must be set to 1 in dpgen calculation
    "kspacing": 0.01  # when KSPACING is set, the above k_points setting becomes␣
↪invalid.
    }
}
```

12.2.2 autotest

This example can be found in examples/dpgen-example/autotest directory.

`dpgen autotest` supports to perform `relaxation`,`eos` (equation of state),`elastic`,`surface`,`vacancy`, and `interstitial` calculations with ABACUS. A `property.json` and `machine.json` file need to be provided. For example,

`property.json`:

```
{
    "structures":     ["confs/"],
    "interaction": {
        "type":          "abacus",
        "incar":         "./INPUT",
        "potcar_prefix":"./",
        "potcars":       {"Al": "Al.PD04.PBE.UPF"},
        "orb_files": {"Al":"Al_gga_10au_100Ry_3s3p2d.orb"}
    },
    "_relaxation": {
            "cal_type": "relaxation",
            "cal_setting":{
                    "input_prop": "./INPUT.rlx"
            }
     },
    "properties": [
        {
         "type":          "eos",
         "vol_start":     0.85,
         "vol_end":       1.15,
         "vol_step":      0.01,
         "cal_setting": {
                         "relax_pos": true,
                         "relax_shape": true,
                         "relax_vol": false,
                         "overwrite_interaction":{
                                     "type": "abacus",
                                     "incar": "./INPUT",
                                     "potcar_prefix":"./",
                                     "orb_files": {"Al":"Al_gga_10au_100Ry_3s3p2d.orb
↪"},
                                     "potcars": {"Al": "Al.PD04.PBE.UPF"} }
                        }
        },
         {
         "type":          "elastic",
         "skip":          false,
         "norm_deform":    1e-2,
         "shear_deform":   1e-2
        },
        {
         "type":          "vacancy",
         "skip":          false,
         "supercell":     [2, 2, 2]
        },
        {
         "type":            "surface",
         "skip":          true,
```

(continues on next page)

(continued from previous page)

```
         "min_slab_size":  15,
         "min_vacuum_size":11,
         "pert_xz":        0.01,
         "max_miller":     3,
         "cal_type":       "static"
        }
        ]
}
```

machine.json

```
{
  "api_version": "1.0",
  "deepmd_version": "2.1.0",
  "train" :[
    {
      "command": "dp",
      "machine": {
        "batch_type": "DpCloudServer",
        "context_type": "DpCloudServerContext",
        "local_root" : "./",
        "remote_profile":{
          "email": "xxx@xxx.xxx",
          "password": "xxx",
          "program_id": 000,
            "input_data":{
                "api_version":2,
                "job_type": "indicate",
                "log_file": "00*/train.log",
                "grouped":true,
                "job_name": "Al-train-VASP",
                "disk_size": 100,
                "scass_type":"c8_m32_1 * NVIDIA V100",
                "platform": "ali",
                "image_name":"LBG_DeePMD-kit_2.1.0_v1",
                "on_demand":0
            }
        }
      },
      "resources": {
        "number_node":123473334635,
        "local_root":"./",
        "cpu_per_node": 4,
        "gpu_per_node": 1,
        "queue_name": "GPU",
        "group_size": 1
      }
    }],
  "model_devi":
    [{
      "command": "lmp -i input.lammps -v restart 0",
      "machine": {
        "batch_type": "DpCloudServer",
        "context_type": "DpCloudServerContext",
        "local_root" : "./",
        "remote_profile":{
          "email": "xxx@xxx.xxx",
```

(continues on next page)

(continued from previous page)

```
        "password": "xxx",
        "program_id": 000,
          "input_data":{
            "api_version":2,
            "job_type": "indicate",
            "log_file": "*/model_devi.log",
            "grouped":true,
            "job_name": "Al-devia-ABACUS",
            "disk_size": 200,
            "scass_type":"c8_m32_1 * NVIDIA V100",
            "platform": "ali",
            "image_name":"LBG_DeePMD-kit_2.1.0_v1",
            "on_demand":0
          }
      }
    },
    "resources": {
      "number_node": 28348383,
      "local_root":"./",
      "cpu_per_node": 4,
      "gpu_per_node": 1,
      "queue_name": "GPU",
      "group_size": 100
    }
  }],
"fp":
  [{
    "command": "OMP_NUM_THREADS=1 mpirun -np 16 abacus",
    "machine": {
      "batch_type": "DpCloudServer",
      "context_type": "DpCloudServerContext",
      "local_root" : "./",
      "remote_profile":{
        "email": "xxx@xxx.xxx",
        "password": "xxx",
       "program_id": 000,
          "input_data":{
            "api_version":2,
            "job_type": "indicate",
            "log_file": "task*/fp.log",
            "grouped":true,
            "job_name": "al-DFT-test",
            "disk_size": 100,
            "scass_type":"c32_m128_cpu",
            "platform": "ali",
            "image_name":"XXXXX",
            "on_demand":0
          }
      }
    },
    "resources": {
      "number_node": 712254638889,
      "cpu_per_node": 32,
      "gpu_per_node": 0,
      "queue_name": "CPU",
      "group_size": 2,
      "local_root":"./",
```

(continues on next page)

(continued from previous page)

```
        "source_list": ["/opt/intel/oneapi/setvars.sh"]
      }
    }
  ]
}
```

For each property, the command `dpgen autotest make property.json` will generate the input files, `dpgen autotest run property.json machine.json` will run the corresponding tasks, and `dpgen autotest post property.json` will collect the final results.

Notes:

- The ABACUS-DPGEN interface can be used in both pw and lcao basis.

12.3 DeepH

DeepH applies meaching learning to predict the Hamiltonian in atomic basis representation. For such purpose, DeepH uses the Hamiltonian and overlap matrices from DFT calculations. Here we introduce how to extract relevant information from ABACUS for the purpose of DeepH training and prediction.

Detailed instructions on installing and running DeepH can be found on its official website. An example for using DeepH with ABACUS is also provided.

Here I intend not to repeat information from the above sources, but to add some minor details related to the setting of ABACUS `INPUT` files.

> Note: Use the LCAO basis for DeepH-related calculations

As mentioned in the README.md file in the above-mentioned example, there are two stages where users need to run ABACUS calculations.

The first stage is during the data preparation phase, where we need to run a series of SCF calculations and output the Hamiltonian and overlap matrices. For such purpose, one needs to add the following line in the `INPUT` file:

```
out_mat_hs2 1
```

Files named data-HR-sparse_SPIN`${x}`.csr and data-SR-sparse_SPIN`${x}`.csr will be generated, which contain the Hamiltonian and overlap matrices respectively in csr format. `${x}` takes value of 0 or 1, based on the spin component. More details on this keyword can be found in the *list of input keywords*.

The second stage is during the inference phase. After DeepH training completes, we can apply the model to predict the Hamiltonian on other systems. For that purpose, we also need the overlap matrices from the new systems, but no SCF calculation is required.

For that purpose, in `INPUT` file we need to make the following specification of the keyword `calculation`:

```
calculation get_S
```

A file named `SR.csr` will be generated in the working directory, which contains the overlap matrix.

12.4 Hefei-NAMD

Hefei-NAMD Non-adiabatic molecular dynamics applies surface hopping to incorporate quantum mechanical effects into molecular dynamics simulations. Surface hopping partially incorporates the non-adiabatic effects by including excited adiabatic surfaces in the calculations, and allowing for 'hops' between these surfaces.

Detailed instructions on installing and running Hefei-NAMD can be found on its official website.

ABACUS provides results of molecular dynamics simulations for Hefei-NAMD to do non-adiabatic molecular dynamics simulations.

The steps are as follows :

1. Add output parameters in INPUT when running MD using ABACUS .

```
out_wfc_lcao    1
out_mat_hs      1
```

Then we obtain output files of hamiltonian matrix, overlap matrix, and wavefunction to do NAMD simulation.

2. Clone Hefei-NAMD codes optimized for ABACUS from website.
3. Modify parameters in `Args.py` including directory of ABACUS output files and NAMD parameters. We can see detailed explanation for all parameters in `Args.py`.
4. Run `NAC.py` to prepare related files for NAMD simulations.

```
sbatch sub_nac
```

5. Run `SurfHop.py` to perform NAMD simulations.

```
sbatch sub_sh
```

And results are under directory namddir in `Args.py`.

12.5 Phonopy

Phonopy is a powerful package to calculate phonon and related properties. The ABACUS interface has been added in Phonopy v.2.19.1. In the following, we take the FCC aluminum as an example:

1. To obtain supercells ($2 \times 2 \times 2$) with displacements, run phonopy:

```
phonopy -d --dim="2 2 2" --abacus
```

2. Calculate forces on atoms in the supercells with displacements. For each SCF calculation, you should specify `stru_file` with `STRU-{number}` and `cal_force=1` in INPUT in order to calculate force using ABACUS. Be careful not to relax the structures

```
echo 'stru_file ./STRU-001' >> INPUT
```

3. Then create 'FORCE_SETS' file using ABACUS inteface:

```
phonopy -f ./disp-{number}/OUT*/running*.log
```

4. Calculate the phonon dispersion:

```
phonopy band.conf --abacus
```

using the following `band.conf` file:

```
ATOM_NAME = Al
DIM = 2 2 2
MESH = 8 8 8
PRIMITIVE_AXES = 0 1/2 1/2  1/2 0 1/2  1/2 1/2 0
BAND= 1 1 1  1/2 1/2 1  3/8 3/8 3/4  0 0 0  1/2 1/2 1/2
BAND_POINTS = 21
BAND_CONNECTION = .TRUE.
```

12.6 Wannier90

Wannier90 is a useful package to generating the maximally-localized Wannier functions (MLWFs), which can be used to compute advanced electronic properties. Some post-processing tools (such as WannierTools, etc.) will use MLWFs for further analysis and calculations.

Currently ABACUS provides an interface to Wannier90 package. The users are assumed to be familiar with the use of Wannier90. The ABACUS-Wannier90 interface is only suitable for nspin=1 or 2, not for nspin=4 or spin-orbit coupling (SOC).

To construct the MLWFs using the wave functions of ABACUS generally requires four steps. Here we use the diamond as an example which can be found in examples/interface_wannier90/.

1. Enter the `ABACUS_towannier90/` folder, prepare a Wannier90 input file `diamond.win`, which is the main input file for Wannier90. Then To generate `diamond.nnkp` file by running Wannier90, which ABACUS will read later:

   ```
   wannier90 -pp diamond.win
   ```

   The content of `diamond.win` is as follows:

   ```
   num_wann        =  4
   num_iter        = 20

   wannier_plot=.true.
   wannier_plot_supercell = 3
   wvfn_formatted = .true.

   begin atoms_frac
   C   -0.12500  -0.1250    -0.125000
   C    0.12500   0.1250     0.125000
   end atoms_frac

   begin projections
   f=0.0,0.0,0.0:s
   f=0.0,0.0,0.5:s
   f=0.0,0.5,0.0:s
   f=0.5,0.0,0.0:s
   end projections

   begin unit_cell_cart
   -1.613990   0.000000   1.613990
    0.000000   1.613990   1.613990
   ```

(continues on next page)

(continued from previous page)

```
-1.613990   1.613990   0.000000
end unit_cell_cart

mp_grid : 4 4 4

begin kpoints
0.0000  0.0000     0.0000
0.0000  0.2500     0.0000
0.0000  0.5000     0.0000
0.0000  0.7500     0.0000
...
end kpoints
```

2. Do a self-consistent calculation and get the converged charge density:

```
cp INPUT-scf INPUT
cp KPT-scf KPT
mpirun -np 4 abacus
```

3. Do a non-self-consistent calculation:

```
cp INPUT-nscf INPUT
cp KPT-nscf KPT
mpirun -np 4 abacus
```

below are the INPUT file (nscf):

```
INPUT_PARAMETERS

ntype                   1
ecutwfc                 50
nbands                  4
calculation             nscf
scf_nmax                50
pw_diag_thr             1.0e-12
scf_thr                 1.0e-15
init_chg                file
symmetry                0
towannier90             1
nnkpfile                diamond.nnkp
```

There are seven interface-related parameters in the `INPUT` file:

- *towannier90*: `1`, generate files for wannier90 code; `0`, do not generate.
- *nnkpfile* : the name of the file generated by running "wannier90 -pp …".
- *wannier_spin*: If you use nspin=2, `up`: calculate the Wannier functions for the spin up components ; `down`: calculate the Wannier functions spin down components.
- *out_wannier_mmn*: control whether to output the "*.mmn" file.
- *out_wannier_amn*: control whether to output the "*.amn" file.
- *out_wannier_eig*: control whether to output the "*.eig" file.
- *out_wannier_unk*: control whether to output the "UNK.*" file.
- *out_wannier_wvfn_formatted*: control what format of the Wannier function file to output, `true`: output the formatted text file; `false`: output the binary file. Note that the `wvfn_formatted` option in `*.win` file

(input file of Wannier90) has to be set accordingly with this option.

Note: You need to turn off the symmetry during the entire nscf calculation.

To setup the `KPT` file according to the `diamond.win` file, which is similar to "begin kpoints ..." in the `diamond.win` file:

```
K_POINTS
64
Direct
0.0000 0.0000 0.0000 0.0156250
0.0000 0.2500 0.0000 0.0156250
0.0000 0.5000 0.0000 0.0156250
0.0000 0.7500 0.0000 0.0156250
...
```

After the nscf calculation, ABACUS will generate `diamond.amn`, `diamond.mmn`, `diamond.eig`, `UNK` files in the `OUT.` folder which are input files needed by Wannier90 code.

4. Copy `.amn`, `.mmn`, `.eig`, `UNK` file to `wannier/` folder, to get the MLWFs by running Wannier90:

```
wannier90 diamond.win
```

Notes:

- The ABACUS-wannier90 interface can be used in both PW and LCAO basis.
- If you want to plot the Wannier function, you must set `wvfn_formatted = .true.` in `diamond.win`, otherwise Wannier90 code cannot read files generated by ABACUS because these files are not binary files. You also have to generate the `diamond.amn` and `UNK` files for the plot. Otherwise, the two types of file are not necessary.

12.7 ASE

12.7.1 Introduction

ASE (Atomic Simulation Environment) provides a set of Python tools for setting, running, and analysing atomic simulations. We have developed an ABACUS calculator (ase-abacus) to be used together with the ASE tools, which exists as an external project with respect to ASE and is maintained by ABACUS developers.

12.7.2 Installation

```
git clone https://gitlab.com/1041176461/ase-abacus.git
cd ase-abacus
python3 setup.py install
```

12.7.3 Environment variables

ABACUS supports two types of basis sets: PW, LCAO. The path of pseudopotential and numerical orbital files can be set throught the environment variables `ABACUS_PP_PATH` and `ABACUS_ORBITAL_PATH`, respectively, e.g.:

```
PP=${HOME}/pseudopotentials
ORB=${HOME}/orbitals
export ABACUS_PP_PATH=${PP}
export ABACUS_ORBITAL_PATH=${ORB}
```

For PW calculations, only `ABACUS_PP_PATH` is needed. For LCAO calculations, both `ABACUS_PP_PATH` and `ABACUS_ORBITAL_PATH` should be set.

12.7.4 ABACUS Calculator

The default initialization command for the ABACUS calculator is

```
from ase.calculators.abacus import Abacus
```

In order to run a calculation, you have to ensure that at least the following parameters are specified, either in the initialization or as environment variables:

keyword	description
`pp`	dict of pseudopotentials for involved elememts, such as `pp={'Al':'Al_ONCV_PBE-1.0.upf',...}`.
`pseu`	directory where the pseudopotential are located, Can also be specified with the `ABACUS_PP_PATH` environment variable. Default: `pseudo_dir=./`.
`basis`	dict of orbital files for involved elememts, such as `basis={'Al':'Al_gga_10au_100Ry_4s4p1d.orb'}`. It must be set if you want to do LCAO calculations. But for pw calculations, it can be omitted.
`basis_`	directory where the orbital files are located, Can also be specified with the `ABACUS_ORBITAL_PATH` environment variable. Default: `basis_dir=./`.
`xc`	which exchange-correlation functional is used. An alternative way to set this parameter is via seting `dft_functional` which is an ABACUS parameter used to specify exchange-correlation functional
`kpts`	a tuple (or list) of 3 integers `kpts=(int, int, int)`, it is interpreted as the dimensions of a Monkhorst-Pack grid, when `kmode` is `Gamma` or `MP`. It is interpreted as k-points, when `kmode` is `Direct`, `Cartesian` or `Line`, and `knumber` should also be set in these modes to denote the number of k-points. Some other parameters for k-grid settings: including `koffset` and `kspacing`.

For more information on pseudopotentials and numerical orbitals, please visit [ABACUS]. The elaboration of input parameters can be found *here*.

The input parameters can be set like::

```
calc = Abacus(profile=profile, ntype=1, ecutwfc=50, scf_nmax=50, smearing_method=
↪'gaussian', smearing_sigma=0.01, basis_type='pw', ks_solver='cg', calculation='scf'␣
↪pp=pp, basis=basis, kpts=kpts)
```

The command to run jobs can be set by specifying `AbacusProfile`::

```
from ase.calculators.abacus import AbacusProfile
abacus = '/usr/local/bin/abacus'
profile = AbacusProfile(argv=['mpirun','-n','2',abacus])
```

in which `abacus` sets the absolute path of the `abacus` executable.

12.7.5 MD Analysis

After molecular dynamics calculations, the log file `running_md.log` can be read. If the 'STRU_MD_*' files are not continuous (e.g. 'STRU_MD_0', 'STRU_MD_5', 'STRU_MD_10'…), the index parameter of read should be as a slice object. For example, when using the command `read('running_md.log', index=slice(0, 15, 5), format='abacus-out')` to parse 'running_md.log', 'STRU_MD_0', 'STRU_MD_5' and 'STRU_MD_10' will be read.

12.7.6 SPAP Analysis

SPAP (Structure Prototype Analysis Package) is written by Dr. Chuanxun Su to analyze symmetry and compare similarity of large amount of atomic structures. The coordination characterization function (CCF) is used to measure structural similarity. An unique and advanced clustering method is developed to automatically classify structures into groups.

If you use this program and method in your research, please read and cite the publication:

`Su C, Lv J, Li Q, Wang H, Zhang L, Wang Y, Ma Y. Construction of crystal structure prototype database: methods and applications. J Phys Condens Matter. 2017 Apr 26;29(16):165901.`

and you should install it first with command `pip install spap`.

12.8 PYATB

12.8.1 Introduction

PYATB (Python ab initio tight binding simulation package) is an open-source software package designed for computing electronic structures and related properties based on the ab initio tight binding Hamiltonian. The Hamiltonian can be directly obtained after conducting self-consistent calculations with ABACUS using numerical atomic orbital (NAO) bases. The package comprises three modules - Bands, Geometric, and Optical, each providing a comprehensive set of tools for analyzing different aspects of a material's electronic structure.

12.8.2 Installation

```
git clone https://github.com/pyatb/pyatb.git
cd pyatb
python setup.py install --record log
```

To customize the `setup.py` file, you must make changes to the **CXX** and **LAPACK_DIR** variables in line with your environment. **CXX** denotes the C++ compiler you intend to use, for instance, icpc (note that it should not be the mpi version). Furthermore, **LAPACK_DIR** is used to specify the Intel MKL path.

After completing the installation process, you can access the `pyatb` executable and corresponding module, which can be imported using the `import pyatb` command.

12.8.3 How to use

We take Bi_2Se_3 as an example to illustrate how to use ABACUS to generate the tight binding Hamiltonian required for PYATB, and then perform calculations related to PYATB functions.

1. Perform ABACUS self consistent calculation:

```
INPUT_PARAMETERS

# System variables
suffix                  Bi2Se3
ntype                   2
calculation             scf
esolver_type            ksdft
symmetry                1
init_chg                atomic

# Plane wave related variables
ecutwfc                 100

# Electronic structure
basis_type              lcao
ks_solver               genelpa
nspin                   4
smearing_method         gauss
smearing_sigma          0.02
mixing_type             broyden
mixing_beta             0.7
scf_nmax                200
scf_thr                 1e-8
lspinorb                1
noncolin                0

# Variables related to output information
out_chg                 1
out_mat_hs2             1
out_mat_r               1
```

After the key parameters `out_mat_hs2` and `out_mat_r` are turned on, ABACUS will generate files containing the Hamiltonian matrix $H(R)$, overlap matrix $S(R)$, and dipole matrix $r(R)$ after completing the self-consistent calculation. These parameters can be found in the ABACUS `INPUT` file.

2. Copy the HR, SR, and rR files output by ABACUS's self-consistent calculation, which are located in the `OUT*` directory and named `data-HR-sparse_SPIN0.csr`, `data-SR-sparse_SPIN0.csr`, and `data-rR-sparse.csr`, respectively. Copy these files to the working directory and write the `Input` file for PYATB:

```
INPUT_PARAMETERS
{
    nspin                           4
    package                         ABACUS
    fermi_energy                    9.557219691497478
    fermi_energy_unit               eV
    HR_route                        data-HR-sparse_SPIN0.csr
    SR_route                        data-SR-sparse_SPIN0.csr
    rR_route                        data-rR-sparse.csr
    HR_unit                         Ry
    rR_unit                         Bohr
```

(continues on next page)

(continued from previous page)

```
    max_kpoint_num                  8000
}

LATTICE
{
    lattice_constant                1.8897162
    lattice_constant_unit           Bohr
    lattice_vector
    -2.069  -3.583614  0.000000
     2.069  -3.583614  0.000000
     0.000   2.389075  9.546667
}

BAND_STRUCTURE
{
    wf_collect                      0
    kpoint_mode                     line
    kpoint_num                      5
    high_symmetry_kpoint
    0.00000 0.00000 0.0000 100  # G
    0.00000 0.00000 0.5000 100  # Z
    0.50000 0.50000 0.0000 100  # F
    0.00000 0.00000 0.0000 100  # G
    0.50000 0.00000 0.0000 1    # L
}
```

For specific input file writing, please refer to PYATB's quick start.

3. Perform PYATB calculation:

```
export OMP_NUM_THREADS=2
mpirun -np 6 pyatb
```

After the calculation is completed, the band structure data and figures of Bi_2Se_3 can be found in the `Out/Band_Structure` folder.

12.9 ShengBTE

12.9.1 Introduction

This tutorial aims to introduce the process of performing density functional theory calculations using ABACUS and calculating lattice thermal conductivity using the ShengBTE software. During the entire calculation process, the following external software are also used: 1) Phonopy for calculating second-order force constants, 2) ASE for converting atomic structures, 3) ShengBTE's thirdorder program for calculating third-order force constants, and 4) finally using ShengBTE to calculate the material's lattice thermal conductivity.

Here is the announcement of ShengBTE with ABACUS: ShengBTE - The ABACUS DFT software can now be used with ShengBTE

Some external packages that need to be combined are mentioned above, and here it is recommended to read the relevant documentation and instructions of these packages:

ShengBTE：https://bitbucket.org/sousaw/shengbte/src/master/

phonopy：http://abacus.deepmodeling.com/en/latest/advanced/interface/phonopy.html

ASE：http://abacus.deepmodeling.com/en/latest/advanced/interface/ase.html

thirdorder: https://bitbucket.org/sousaw/thirdorder/src/master/

12.9.2 Prepare

The ABACUS software package provides an example of using ABACUS+ShengBTE to calculate the lattice thermal conductivity in the `examples/interface_ShengBTE` folder. The example includes two folders: `LCAO` (Linear Combination of Atomic Orbitals) which uses numerical atomic orbitals and `PW` (Plane wave) which uses plane wave basis vectors. Each folder contains three subfolders: `2nd`, `3rd`, and `shengbte`, which respectively store the relevant files for calculating second-order force constants using Phonopy (`2nd`), third-order force constants using the thirdorder program (`3rd`), and lattice thermal conductivity using ShengBTE (`shengbte`).

12.9.3 How to use

Taking the `LCAO` folder as an example, we provide the test case of a diamond structure Si structure containing 2 atoms with the norm-conserving pseudopotential `Si_ONCV_PBE-1.0.upf` and the atomic orbital file `Si_gga_7au_100Ry_2s2p1d.orb` (GGA functional, 7 a.u. cut-off radius, 100 Ry energy cut-off, and DZP orbitals containing 2s2p1d).

1. Calculating the second-order force constants

It would be best to combine Phonopy and ASE with ABACUS to calculate the second-order force constants. First, enter the `2nd` folder.

1.1 Structure optimization

Before performing lattice thermal conductivity calculations, it is necessary to optimize the atomic configuration of the simulated material system. The following is the atomic configuration file `STRU` obtained after structure optimization (relax) using ABACUS. In this example, for simplicity, a 2*2*2 Brillouin zone k-point sampling was used in the structure optimization process, with an energy cutoff value of 100 Ry for plane waves (the plane wave basis vector is also used in LCAO). Note that the actual calculation should use more convergent k-point sampling.

```
ATOMIC_SPECIES
Si 28.0855 Si_ONCV_PBE-1.0.upf

NUMERICAL_ORBITAL
Si_gga_7au_100Ry_2s2p1d.orb

LATTICE_CONSTANT
1.88972612546

LATTICE_VECTORS
0 2.81594778072 2.81594778072 #latvec1
2.81594778072 0 2.81594778072 #latvec2
2.81594778072 2.81594778072 0 #latvec3

ATOMIC_POSITIONS
Direct # direct coordinate

Si #label
0 #magnetism
```

(continues on next page)

(continued from previous page)

```
2 #number of atoms
0.875  0.875  0.875  m  0  0  0
0.125  0.125  0.125  m  0  0  0
```

1.2 Calculating the second-order force constants

The Phonopy software is called to generate multiple atomic configurations of the supercell and corresponding perturbations needed for calculation with the following command:

```
phonopy setting.conf --abacus -d
```

where the `setting.conf` file reads:

```
DIM = 2 2 2
ATOM_NAME = Si
```

In this Si example, we only need to generate one perturbed atomic configuration, `STRU-001`. Perform SCF calculations (SCF stands for Self-Consistent Field and represents the iterative self-consistent calculation of density functional theory) on all perturbed configurations (in this case, there is only one for Si) to obtain the forces on the atoms. Afterward, use the following command to generate the `FORCE_SET` file:

```
phonopy -f OUT.DIA-50/running_scf.log
```

Tip: In the input file `INPUT` of ABACUS, you can set the variable `stru_file`, which corresponds to the atomic configuration file `STRU-001`, and ABACUS will read the structure file directly.

Next, set the `band.conf` file to calculate the phonon spectrum and the second-order force constants:

```
phonopy -p band.conf --abacus
```

The `band.conf` file mentioned here contains the following contents (you can refer to the Phonopy documentation for specific parameter meanings):

```
ATOM_NAME = Si
DIM = 2 2 2
MESH = 8 8 8
PRIMITIVE_AXES = 1 0 0 0 1 0 0 0 1
BAND = 0.0 0.0 0.0  0.5 0.0 0.5  0.625  0.25  0.625, 0.375 0.375 0.75  00 0.0 0.0  0.
↪5 0.5 0.5
BAND_POINTS = 101
BAND_CONNECTION = .TRUE.
FORCE_CONSTANTS = WRITE
FULL_FORCE_CONSTANTS = .TRUE.
```

After this step, the Phonopy software will generate `band.yaml` (for plotting the phonon spectrum) and the `FORCE_CONSTANTS` file. The data contained in the `FORCE_CONSTANTS` file is the second-order force constants. It is important to set `FULL_FORCE_CONSTANTS = .TRUE.`, which outputs all the second-order force constants. Otherwise, there may be errors when ShengBTE reads the data.

In addition, you can use the following command to output the gnuplot format of the phonon spectrum for plotting:

```
phonopy-bandplot --gnuplot > pho.dat
```

1.3 Post-processing

Note that ShengBTE software requires the unit of the data in the FORCE_CONSTANTS_2ND file to be eV/Å^2, but the unit of the FORCE_CONSTANTS calculated by ABACUS combined with Phonopy is eV/(Å*au), where au is the atomic unit system and 1 au = 0.52918 Å. You can use the provided au2si.py script in the 2nd directory to convert the units and generate the FORCE_CONSTANTS_2ND file. The command is as follows:

```
python au2si.py
```

The FORCE_CONSTANTS_2ND file is provided in the shengbte folder for reference to the calculation results.

2. Calculating the third-order force constants

To calculate the third-order force constants, you need to combine with the thirdorder program and output the third-order force constant file FORCE_CONSTANTS_3RD. However, thirdorder currently only supports reading input and output files from VASP and QE. Therefore, we are using thirdorder by converting ABACUS's structure and output force files to POSCAR and vasprun.xml, respectively. Please enter the 3rd folder first, and the specific steps will be described below.

2.1 Obtaining perturbed configurations

First, convert the optimized STRU file from ABACUS software to POSCAR (the converted POSCAR file is already provided in the directory, or you can do this conversion by yourself).

Then, run the thirdorder_vasp.py program to generate a series of atomic configuration files 3RD.POSCAR.* after perturbation. For example, in this example, a total of 40 configurations were generated:

```
thirdorder_vasp.py sow 2 2 2 -2
```

Run pos2stru.py to convert the above POSCAR to STRU file. Note that this script calls functions from the ASE software package (ASE needs to be installed in advance):

```
python pos2stru.py
```

Note: The dpdata software cannot be called here to perform the conversion. This is because the dpdata software forces the lattice to change into a lower triangular matrix, which is equivalent to rotating the lattice and leads to a corresponding rotation in the direction of the interatomic forces, which will cause errors.

2.2 Calculation of atomic forces for perturbation configurations

You can refer to the run_stru.sh script provided in the directory to batch generate SCF-* folders and submit calculations. Here, ABACUS needs to perform SCF calculations on 40 atomic configurations, which may take some time. It is recommended to run each SCF separately in the SCF-* folder. The scf_thr parameter in INPUT file should be set to at least 1e-8 to obtain converged results.

After the calculations are complete, run aba2vasp.py to package the atomic forces calculated by ABACUS into the vasprun.xml format and place them in each SCF-* folder with the following command:

```
python aba2vasp.py
```

The vasprun.xml format is illustrated as follows:

```
<modeling>
    <calculation>
        <varray name="forces">
            <v>1.865e-05 -0.04644196 -0.00153852</v>
            <v>-1.77e-05 -0.00037715 -0.00149635</v>
            <v>1.973e-05 0.002213 -0.00149461</v>
            <v>-1.976e-05 0.00065303 -0.0014804</v>
            <v>8.31e-06 -0.0003306 -0.00024288</v>
            <v>-8.25e-06 -0.00038306 -0.00025385</v>
            <v>1.071e-05 0.00060621 -0.00025797</v>
            <v>-1.05e-05 -0.00014553 -0.00027532</v>
            <v>0.00668053 0.00645634 -0.04642593</v>
            <v>-0.00668085 0.00645595 -0.00040122</v>
            <v>-0.00650454 0.00628877 -0.00025123</v>
            <v>0.00650504 0.00628892 -0.00028948</v>
            <v>-0.00039591 2.479e-05 0.00223371</v>
            <v>0.00039608 2.426e-05 0.0006732</v>
            <v>0.0003264 3.122e-05 0.00052874</v>
            <v>-0.00032589 3.415e-05 -0.00023577</v>
            <v>-2.908e-05 -0.00832477 0.00635709</v>
            <v>3.737e-05 -0.00125057 -7.444e-05</v>
            <v>-2.582e-05 0.00656076 0.00636285</v>
            <v>2.566e-05 -0.00049974 -6.661e-05</v>
            <v>-5.431e-05 0.00502637 0.00639077</v>
            <v>4.553e-05 -0.00180978 0.0001325</v>
            <v>-3.609e-05 -0.00676473 0.00638092</v>
            <v>3.806e-05 5.503e-05 0.00012759</v>
            <v>-0.00670704 0.00646596 0.01310437</v>
            <v>0.00670119 3.673e-05 0.00602948</v>
            <v>0.00036366 0.00627899 -0.00657272</v>
            <v>-0.00036508 2.288e-05 0.00026009</v>
            <v>0.00648649 0.0064463 -0.00036521</v>
            <v>-0.00648098 1.594e-05 0.00671469</v>
            <v>-0.00034493 0.00630074 0.00662932</v>
            <v>0.00034331 4.157e-05 -0.0002028</v>
        </varray>
    </calculation>
</modeling>
```

Finally, execute the following command:

```
find SCF-* -name vasprun.xml|sort -n|thirdorder_vasp.py reap 2 2 2 -2
```

Then, the third-order force constant file `FORCE_CONSTANTS_3RD` can be obtained by running the above command. The `FORCE_CONSTANTS_3RD` file is provided in the shengbte folder for reference in calculating the results.

3. Run ShengBTE to obtain lattice thermal conductivity

Enter the `shengbte` folder, in which the three files `CONTROL` (parameter file of ShengBTE), `FORCE_CONSTANTS_2ND` (second-order force constant file), and `FORCE_CONSTANTS_3RD` (third-order force constant file) have been prepared. Next, run ShengBTE with the following command to obtain the lattice thermal conductivity, where the calculation results are given in the Ref folder for reference:

```
mpirun -n 10 ShengBTE
```

12.9.4 Conclusion

For using plane wave (PW) in ABACUS to perform ShengBTE calculations, similar procedures should be followed. However, the `scf_thr` parameter in the `INPUT` file for calculating the third-order force constant needs to be set to at least 1e-12. The experimental lattice thermal conductivity of Si at 300 K is around 150 W/(m K), while the calculated thermal conductivity of Si at 300 K is around 100 W/(m K) by using the provided example. This is because, as a demo, a 2*2*2 expanded cell and a 2*2*2 K-point are used in the example, but the results are not converged yet with respect to the given system size and k-points. In actual research, the size of the supercell and the sampling scheme of K-points need to be tested to obtain converged results.

12.10 CANDELA

CANDELA is short for Collection of ANalysis DEsigned for Large-scale Atomic simulations. It is developed by MCresearch to conduct analyses on MD trajectory in different formats. Right now the program only supports analysis of pair distribution function (PDF), static structure factor (SSF) and mean square displacement (MSD). The minimum supported version of ABACUS is 3.2.0.

12.10.1 Requirements for using CANDELA

For Detailed usage of CANDELA, please refer to the official document. There are two things which need special attention in using CANDELA with ABACUS. First, the input file of CANDELA only takes the name of `INPUT`, the same as ABACUS input file, so you should not run CANDELA in the same folder where you run ABACUS. Second, to use CANDELA to postprocess ABACUS MD trajectory, the following parameters have to be specified in the `INPUT` file of CANDELA in addition to other required parameters:

1. `geo_in_type` has to be set to `ABACUS`;
2. `msd_dt` has to be specified in unit of picosecond, especially in the case of `calculation` = `msd`;
3. `geo_directory` has to be set to the path to the `MD_dump` file in the `OUT.xxx` folder. As a result, a CANDELA `INPUT` file for calculating PDF from ABACUS should be something like this:

```
calculation  pdf # Pair Distribution Function.
system Al
geo_in_type  LAMMPS
geo_directory ../geo/Al64.dump
geo_1        0
geo_2        20
geo_interval 1
geo_ignore   4

geo_out      pdf.txt # output pdf name.

ntype        1        # number of different types of atoms.
natom        64   # total number of atoms.
natom1       64
rcut         6
dr           0.01     # delta r in real space

struf_dgx   0.05
struf_ng    200
```

More examples of CANDELA `INPUT` with ABACUS can be found in the test directory.

12.11 TB2J

12.11.1 Introduction

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 Wannier functions 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. The ABACUS interface has been added since TB2J version 0.8.0.

For more information, see the documentation on https://tb2j.readthedocs.io/en/latest/

12.11.2 Installation

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

```
pip install TB2J
```

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

```
git clone https://github.com/mailhexu/TB2J.git
cd TB2J
python setup.py install
```

12.11.3 The Heisenberg model

The Heisenberg Hamiltonian in TB2J contains three different parts, which are:

$$E = -\sum_{i \neq j} \left[J_{\text{iso}}^{ij} \vec{S}_i \cdot \vec{S}_j + \vec{S}_i J_{\text{ani}}^{ij} \vec{S}_j + \vec{D}_{ij} \cdot (\vec{S}_i \times \vec{S}_j) \right],$$

where J_{iso}^{ij} represents the isotropic exchange, J_{ani}^{ij} represents the symmetric anisotropic exhcange which is a 3×3 tensor with $J^{\text{ani}} = J^{\text{ani,T}}$, $\vec{D}_{ij}$ represents the Dzyaloshinskii-Moriya interaction (DMI).

> **Note:** Exchange parameters conventions for other Heisenberg Hamiltonian can be found in Conventions of Heisenberg Model.

12.11.4 How to use

With the LCAO basis set, TB2J can directly take the output and compute the exchange parameters. For the PW and LCAO-in-PW basis set, the Wannier90 interace can be used instead. In this tutorial we will use LCAO.

Collinear calculation without SOC

We take Fe as an example to illustrate how to use ABACUS to generate the input files required for TB2J.

1. Perform ABACUS calculation.

`INPUT` file:

```
INPUT_PARAMETERS
#Parameters (1.General)
suffix                          Fe
stru_file                       STRU
kpoint_file                     KPT
pseudo_dir                      ./
orbital_dir                     ./
calculation                     scf
ntype                           1
nspin                           2
symmetry                        1
noncolin                        0
lspinorb                        0

#Parameters (2.PW)
ecutwfc                         100
scf_thr                         1.0e-6
init_chg                        atomic
out_mul                         1

#Parameters (4.Relaxation)
ks_solver                       genelpa
scf_nmax                        200
out_bandgap                     0

#Parameters (5.LCAO)
basis_type                      lcao
gamma_only                      0

#Parameters (6.Smearing)
smearing_method                 gauss
smearing_sigma                  0.001

#Parameters (7.Charge Mixing)
mixing_type                     broyden
mixing_beta                     0.2

# Variables related to output information
out_mat_hs2                     1
```

`STRU` file:

```
ATOMIC_SPECIES
Fe  55.845         Fe_ONCV_PBE_FR-1.0.upf

NUMERICAL_ORBITAL
Fe_gga_8au_100Ry_4s2p2d1f.orb

LATTICE_CONSTANT
1.8897261258369282

LATTICE_VECTORS
2.8660000000      0.0000000000      0.0000000000
```

(continues on next page)

(continued from previous page)

```
0.0000000000      2.8660000000      0.0000000000
0.0000000000      0.0000000000      2.8660000000

ATOMIC_POSITIONS
Direct

Fe
5.0000000000
2
0.0000000000 0.0000000000 0.0000000000 1 1 1 mag 2.5
0.5000000000 0.5000000000 0.5000000000 1 1 1 mag 2.5
```

and `KPT` file:

```
K_POINTS
0
Gamma
8 8 8 0 0 0
```

After the key parameter `out_mat_hs2` is turned on, 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:

```
suffix                          Fe
```

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

2. Perform TB2J calculation:

```
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 $7 \times 7 \times 7$ k-point grid. This allows for the calculation of exchange between spin pairs between $7 \times 7 \times 7$ 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.

The description of the output files in `TB2J_results` can be found in the TB2J documentation. We can find the exchange parameters with `Fe` by :

```
grep "Fe" exchange.out
```

the following contents showing the first, second and third nearest neighbor exchange parameters as 18.6873, 9.9213 and 0.8963 meV, resoectively. More equivalent exchange parameters are also shown.

```
   Fe1   Fe2   ( -1,  -1,  -1) 18.6873   (-1.433, -1.433, -1.433)  2.482
   Fe1   Fe2   ( -1,  -1,   0) 18.6867   (-1.433, -1.433,  1.433)  2.482
   Fe1   Fe2   ( -1,   0,  -1) 18.6866   (-1.433,  1.433, -1.433)  2.482
   Fe1   Fe2   ( -1,   0,   0) 18.6873   (-1.433,  1.433,  1.433)  2.482
   Fe1   Fe2   (  0,  -1,  -1) 18.6873   ( 1.433, -1.433, -1.433)  2.482
   Fe1   Fe2   (  0,  -1,   0) 18.6866   ( 1.433, -1.433,  1.433)  2.482
```

(continues on next page)

(continued from previous page)

```
  Fe1   Fe2   (  0,   0,  -1) 18.6867   ( 1.433,  1.433, -1.433)  2.482
  Fe1   Fe2   (  0,   0,   0) 18.6873   ( 1.433,  1.433,  1.433)  2.482
  Fe2   Fe1   (  0,   0,   0) 18.6873   (-1.433, -1.433, -1.433)  2.482
  Fe2   Fe1   (  0,   0,   1) 18.6867   (-1.433, -1.433,  1.433)  2.482
  Fe2   Fe1   (  0,   1,   0) 18.6866   (-1.433,  1.433, -1.433)  2.482
  Fe2   Fe1   (  0,   1,   1) 18.6873   (-1.433,  1.433,  1.433)  2.482
  Fe2   Fe1   (  1,   0,   0) 18.6873   ( 1.433, -1.433, -1.433)  2.482
  Fe2   Fe1   (  1,   0,   1) 18.6866   ( 1.433, -1.433,  1.433)  2.482
  Fe2   Fe1   (  1,   1,   0) 18.6867   ( 1.433,  1.433, -1.433)  2.482
  Fe2   Fe1   (  1,   1,   1) 18.6873   ( 1.433,  1.433,  1.433)  2.482
  Fe1   Fe1   ( -1,   0,   0)  9.9213   (-2.866,  0.000,  0.000)  2.866
  Fe2   Fe2   ( -1,   0,   0)  9.9213   (-2.866,  0.000,  0.000)  2.866
  Fe1   Fe1   (  0,  -1,   0)  9.9211   ( 0.000, -2.866,  0.000)  2.866
  Fe2   Fe2   (  0,  -1,   0)  9.9211   ( 0.000, -2.866,  0.000)  2.866
  Fe1   Fe1   (  0,   0,  -1)  9.9210   ( 0.000,  0.000, -2.866)  2.866
  Fe2   Fe2   (  0,   0,  -1)  9.9210   ( 0.000,  0.000, -2.866)  2.866
  Fe1   Fe1   (  0,   0,   1)  9.9210   ( 0.000,  0.000,  2.866)  2.866
  Fe2   Fe2   (  0,   0,   1)  9.9210   ( 0.000,  0.000,  2.866)  2.866
  Fe1   Fe1   (  0,   1,   0)  9.9211   ( 0.000,  2.866,  0.000)  2.866
  Fe2   Fe2   (  0,   1,   0)  9.9211   ( 0.000,  2.866,  0.000)  2.866
  Fe1   Fe1   (  1,   0,   0)  9.9213   ( 2.866,  0.000,  0.000)  2.866
  Fe2   Fe2   (  1,   0,   0)  9.9213   ( 2.866,  0.000,  0.000)  2.866
  Fe1   Fe1   ( -1,  -1,   0)  0.8963   (-2.866, -2.866,  0.000)  4.053
  Fe2   Fe2   ( -1,  -1,   0)  0.8963   (-2.866, -2.866,  0.000)  4.053
  Fe1   Fe1   ( -1,   0,  -1)  0.8970   (-2.866,  0.000, -2.866)  4.053
  Fe2   Fe2   ( -1,   0,  -1)  0.8970   (-2.866,  0.000, -2.866)  4.053
```

Several other formats of the exchange parameters are also provided in the `TB2J_results` directory , which can be used in spin dynamics code, e.g. MULTIBINIT, Vampire.

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.

Parameters of abacus2J.py

We can use the command

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

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

Acknowledgments

We thanks to Xu He to provide critical interface support.

CHAPTER

THIRTEEN

DETAILED INTRODUCTION OF THE INPUT FILES

13.1 Full List of INPUT Keywords

back to top

13.1.1 System variables

These variables are used to control general system parameters.

suffix

- **Type**: String
- **Description**: In each run, ABACUS will generate a subdirectory in the working directory. This subdirectory contains all the information of the run. The subdirectory name has the format: OUT.suffix, where the `suffix` is the name you can pick up for your convenience.
- **Default**: ABACUS

ntype

- **Type**: Integer
- **Description**: Number of different atom species in this calculation. If this value is not equal to the atom species in the STRU file, ABACUS will stop and quit. If not set or set to 0, ABACUS will automatically set it to the atom species in the STRU file.
- **Default**: 0

calculation

- **Type**: String
- **Description**: Specify the type of calculation.
  - scf: do self-consistent electronic structure calculation
  - relax: do structure relaxation calculation, one can use `relax_nmax` to decide how many ionic relaxations you want
  - cell-relax: do variable-cell relaxation calculation
  - nscf: do the non self-consistent electronic structure calculations. For this option, you need a charge density file. For nscf calculations with planewave basis set, pw_diag_thr should be <= 1e-3
  - get_pchg: For LCAO basis. Please see the explanation for variable `nbands_istate` and `bands_to_print`
  - get_wf: Envelope function for LCAO basis. Please see the explanation for variable `nbands_istate`
  - md: molecular dynamics
  - test_memory : checks memory required for the calculation. The number is not quite reliable, please use it with care
  - test_neighbour : only performs neighbouring atom search, please specify a positive *search_radius* manually.
  - gen_bessel : generates projectors (a series of Bessel functions) for DeePKS; see also keywords bessel_descriptor_lmax, bessel_descriptor_rcut and bessel_descriptor_tolerence. A file named `jle.orb` will be generated which contains the projectors. An example is provided in examples/H2O-deepks-pw

  - get_S : only works for multi-k calculation with LCAO basis. Generates and writes the overlap matrix to a file named `SR.csr` in the working directory. The format of the file will be the same as that generated by *out_mat_hs2*

- **Default**: scf

esolver_type

- **Type**: String
- **Description**: choose the energy solver.
  - ksdft: Kohn-Sham density functional theory
  - ofdft: orbital-free density functional theory
  - sdft: *stochastic density functional theory*
  - tddft: real-time time-dependent density functional theory (TDDFT)
  - lj: Leonard Jones potential
  - dp: DeeP potential, see details in *md.md*
- **Default**: ksdft

symmetry

- **Type**: Integer
- **Description**: takes value 1, 0 or -1.
  - -1: No symmetry will be considered.
  - 0: Only time reversal symmetry would be considered in symmetry operations, which implied k point and -k point would be treated as a single k point with twice the weight.
  - 1: Symmetry analysis will be performed to determine the type of Bravais lattice and associated symmetry operations. (point groups, space groups, primitive cells, and irreducible k-points)
- **Default**:
  - 0:
    - if calculation==*md/nscf/get_pchg/get_wf/get_S* or *[gamma_only]==True*;
    - If (dft_fuctional==*hse/hf/pbe0/scan0/opt_orb* or rpa==*True*). Currently *symmetry==1* is not supported in EXX (exact exchange) calculation.
  - 1: else

symmetry_prec

- **Type**: Real
- **Description**: The accuracy for symmetry judgment. Usually the default value is good enough, but if the lattice parameters or atom positions in STRU file is not accurate enough, this value should be enlarged.

  Note: if calculation==*cell_relax*, this value can be dynamically changed corresponding to the variation of accuracy of the lattice parameters and atom positions during the relaxation. The new value will be printed in `OUT.${suffix}/running_cell-relax.log` in that case.
- **Default**: 1.0e-6
- **Unit**: Bohr

symmetry_autoclose

- **Type**: Boolean
- **Availability**: symmetry==*1*
- **Description**: Control how to deal with error in symmetry analysis due to inaccurate lattice parameters or atom positions in STRU file, especially useful when calculation==*cell-relax*
  - False: quit with an error message
  - True: automatically set symmetry to 0 and continue running without symmetry analysis
- **Default**: True

kpar

- **Type**: Integer
- **Description**: divide all processors into kpar groups, and k points will be distributed among each group. The value taken should be less than or equal to the number of k points as well as the number of MPI processes.
- **Default**: 1

bndpar

- **Type**: Integer
- **Description**: divide all processors into bndpar groups, and bands (only stochastic orbitals now) will be distributed among each group. It should be larger than 0.
- **Default**: 1

latname

- **Type**: String
- **Description**: Specifies the type of Bravias lattice. When set to `none`, the three lattice vectors are supplied explicitly in STRU file. When set to a certain Bravais lattice type, there is no need to provide lattice vector, but a few lattice parameters might be required. For more information regarding this parameter, consult the *page on STRU file*.

  Available options are (correspondence with ibrav in QE(Quantum Espresso) is given in parenthesis):

  - none: free structure
  - sc: simple cubic (1)
  - fcc: face-centered cubic (2)
  - bcc: body-centered cubic (3)
  - hexagonal: hexagonal (4)
  - trigonal: trigonal (5)
  - st: simple tetragonal (6)
  - bct: body-centered tetragonal (7)
  - so: orthorhombic (8)
  - baco: base-centered orthorhombic (9)
  - fco: face-centered orthorhombic (10)
  - bco: body-centered orthorhombic (11)
  - sm: simple monoclinic (12)
  - bacm: base-centered monoclinic (13)
  - triclinic: triclinic (14)
- **Default**: none

psi_initializer

- **Type**: Integer
- **Description**: enable the experimental feature psi_initializer, to support use numerical atomic orbitals initialize wavefunction (`basis_type pw` case).

  NOTE: this feature is not well-implemented for `nspin 4` case (closed presently), and cannot use with `calculation nscf`/`esolver_type sdft` cases. Available options are:

  - 0: disable psi_initializer
  - 1: enable psi_initializer
- **Default**: 0

init_wfc

- **Type**: String
- **Description**: Only useful for plane wave basis only now. It is the name of the starting wave functions. In the future. we should also make this variable available for localized orbitals set.

  Available options are:

  - atomic: from atomic pseudo wave functions. If they are not enough, other wave functions are initialized with random numbers.
  - atomic+random: add small random numbers on atomic pseudo-wavefunctions
  - file: from binary files `WAVEFUNC*.dat`, which are output by setting *out_wfc_pw* to `2`.
  - random: random numbers

  with `psi_initializer 1`, two more options are supported:

  - nao: from numerical atomic orbitals. If they are not enough, other wave functions are initialized with random numbers.
  - nao+random: add small random numbers on numerical atomic orbitals
- **Default**: atomic

init_chg

- **Type**: String
- **Description**: This variable is used for both plane wave set and localized orbitals set. It indicates the type of starting density.
  - atomic: the density is starting from the summation of the atomic density of single atoms.
  - file: the density will be read in from a binary file `charge-density.dat` first. If it does not exist, the charge density will be read in from cube files. Besides, when you do `nspin=1` calculation, you only need the density file SPIN1_CHG.cube. However, if you do `nspin=2` calculation, you also need the density file SPIN2_CHG.cube. The density file should be output with these names if you set out_chg = 1 in INPUT file.
- **Default**: atomic

init_vel

- **Type**: Boolean
- **Description**:
  - True: read the atom velocity (atomic unit : 1 a.u. = 21.877 Angstrom/fs) from the atom file (`STRU`) and determine the initial temperature *md_tfirst*. If *md_tfirst* is unset or less than zero, `init_vel` is autoset to be `true`.
  - False: assign value to atom velocity using Gaussian distributed random numbers.
- **Default**: False

nelec

- **Type**: Real
- **Description**:
  - 0.0: the total number of electrons will be calculated by the sum of valence electrons (i.e. assuming neutral system).
  - `>0.0`: this denotes the total number of electrons in the system. Must be less than 2*nbands.
- **Default**: 0.0

nelec_delta

- **Type**: Real
- **Description**: the total number of electrons will be calculated by `nelec`+`nelec_delta`.
- **Default**: 0.0

nupdown

- **Type**: Real
- **Description**:
  - 0.0: no constrain apply to system.
  - `>0.0`: this denotes the difference number of electrons between spin-up and spin-down in the system. The range of value must in [-nelec ~ nelec]. It is one method of constraint DFT, the fermi energy level will separate to E_Fermi_up and E_Fermi_down.
- **Default**: 0.0

dft_functional

- **Type**: String
- **Description**: In our package, the XC functional can either be set explicitly using the `dft_functional` keyword in `INPUT` file. If `dft_functional` is not specified, ABACUS will use the xc functional indicated in the pseudopotential file. On the other hand, if dft_functional is specified, it will overwrite the functional from pseudopotentials and performs calculation with whichever functional the user prefers. We further offer two ways of supplying exchange-correlation functional. The first is using 'short-hand' names such as 'LDA', 'PBE', 'SCAN'. A complete list of 'short-hand' expressions can be found in the source code. The other way is only available when ***compiling with LIBXC***, and it allows for supplying exchange-correlation functionals as combinations of LIBXC keywords for functional components, joined by a plus sign, for example, 'dft_functional='LDA_X_1D_EXPONENTIAL+LDA_C_1D_CSC'. The list of LIBXC keywords can be found on its website. In this way, **we support all the LDA,GGA and mGGA functionals provided by LIBXC**.

  Furthermore, the old INPUT parameter exx_hybrid_type for hybrid functionals has been absorbed into dft_functional. Options are `hf` (pure Hartree-Fock), `pbe0`(PBE0), `hse` (Note: in order to use HSE functional, LIBXC is required). Note also that HSE has been tested while PBE0 has NOT been fully tested yet, and the maximum CPU cores for running exx in parallel is $N(N+1)/2$, with N being the number of atoms. And forces for hybrid functionals are not supported yet.

  If set to `opt_orb`, the program will not perform hybrid functional calculation. Instead, it is going to generate opt-ABFs as discussed in this article.

- **Default**: same as UPF file.

xc_temperature

- **Type**: Real
- **Description**: specifies temperature when using temperature-dependent XC functionals (KSDT and so on).
- **Default** : 0.0
- **Unit**: Ry

pseudo_rcut

- **Type**: Real
- **Description**: Cut-off of radial integration for pseudopotentials
- **Default**: 15
- **Unit**: Bohr

pseudo_mesh

- **Type**: Integer
- **Description**:
  - 0: use our own mesh for radial integration of pseudopotentials
  - 1: use the mesh that is consistent with quantum espresso
- **Default**: 0

mem_saver

- **Type**: Boolean
- **Description**: Used only for nscf calculations.
  - 0: no memory saving techniques are used.
  - 1: a memory saving technique will be used for many k point calculations.
- **Default**: 0

diago_proc

- **Type**: Integer
- **Availability**: pw base
- **Description**:
  - 0: it will be set to the number of MPI processes. Normally, it is fine just leave it to the default value.
  - `>0`: it specifies the number of processes used for carrying out diagonalization. Must be less than or equal to total number of MPI processes. Also, when cg diagonalization is used, diago_proc must be the same as the total number of MPI processes.

- **Default**: 0

nbspline

- **Type**: Integer
- **Description**: If set to a natural number, a Cardinal B-spline interpolation will be used to calculate Structure Factor. `nbspline` represents the order of B-spline basis and a larger one can get more accurate results but cost more. It is turned off by default.
- **Default**: -1

kspacing

- **Type**: Real
- **Description**: Set the smallest allowed spacing between k points, unit in 1/bohr. It should be larger than 0.0, and suggest smaller than 0.25. When you have set this value > 0.0, then the KPT file is unnecessary, and the number of K points nk_i = max(1, int(|b_i|/KSPACING_i)+1), where b_i is the reciprocal lattice vector. The default value 0.0 means that ABACUS will read the applied KPT file. If only one value is set (such as `kspacing 0.5`), then kspacing values of a/b/c direction are all set to it; and one can also set 3 values to set the kspacing value for a/b/c direction separately (such as: `kspacing 0.5 0.6 0.7`).

  Note: if gamma_only is set to be true, kspacing is invalid.
- **Default**: 0.0

min_dist_coef

- **Type**: Real
- **Description**: a factor related to the allowed minimum distance between two atoms. At the beginning, ABACUS will check the structure, and if the distance of two atoms is shorter than min_dist_coef*(standard covalent bond length), we think this structure is unreasonable. If you want to calculate some structures in extreme conditions like high pressure, you should set this parameter as a smaller value or even 0.
- **Default**: 0.2

device

- **Type**: String
- **Description**: Specifies the computing device for ABACUS.

  Available options are:

  - cpu: for CPUs via Intel, AMD, or Other supported CPU devices
  - gpu: for GPUs via CUDA or ROCm.

  Known limitations:

  - pw basis: required by the `gpu` acceleration options
  - cg/bpcg/dav ks_solver: required by the `gpu` acceleration options
- **Default**: cpu

precision

- **Type**: String
- **Description**: Specifies the precision of the PW_SCF calculation.

  Available options are:

  - single: single precision
  - double: double precision

  Known limitations:

  - pw basis: required by the `single` precision options
  - cg/bpcg/dav ks_solver: required by the `single` precision options
- **Default**: double

back to top

13.1.2 Variables related to input files

These variables are used to control parameters related to input files.

stru_file

- **Type**: String
- **Description**: the name of the structure file
  - Containing various information about atom species, including pseudopotential files, local orbitals files, cell information, atom positions, and whether atoms should be allowed to move.
  - Refer to Doc
- **Default**: STRU

kpoint_file

- **Type**: String
- **Description**: the name of the k-points file
  - In atomic orbitals basis with `gamma_only` set to true, the `KPT` file is unnecessary, because a `KPT` file will be generated automatically.
  - When more than one k-points are required, an explicit `KPT` file is mandatory.
  - Refer to Doc
- **Default**: KPT

pseudo_dir

- **Type**: String
- **Description**: the pseudopotential file directory
  - This parameter is combined with the pseudopotential filenames in the STRU file to form the complete pseudopotential file paths.
  - Example: set pseudo_dir to "…/" with "Si.upf" which specified under "ATOMIC_SPECIES" in STRU file, ABACUS will open the pseudopotential file in path "…/Si.upf".
- **Default**: ""

orbital_dir

- **Type**: String
- **Description**: the orbital file directory
  - This parameter is combined with orbital filenames in the STRU file to form the complete orbital file paths.
  - Example: set orbital_dir to "…/" with "Si.orb" which specified under "NUMERICAL_ORBITAL" in STRU file, ABACUS will open the orbital file in path "…/Si.orb".
- **Default**: ""

read_file_dir

- **Type**: String
- **Description**: Indicates the location of files, such as electron density (`SPIN1_CHG.cube`), required as a starting point.
  - Example: './' implies the files to be read are located in the working directory.
- **Default**: OUT.$suffix

wannier_card

- **Type**: String
- **Availability**: Using ABACUS with Wannier90.
- **Description**: The name of the input file related to Wannier90.
- **Default**: "none"

back to top

13.1.3 Plane wave related variables

These variables are used to control the plane wave related parameters.

ecutwfc

- **Type**: Real
- **Description**: Energy cutoff for plane wave functions, the unit is **Rydberg**. Note that even for localized orbitals basis, you still need to setup an energy cutoff for this system. Because our local pseudopotential parts and the related force are calculated from plane wave basis set, etc. Also, because our orbitals are generated by matching localized orbitals to a chosen set of wave functions from a certain energy cutoff, this set of localize orbitals is most accurate under this same plane wave energy cutoff.
- **Default**: 50

ecutrho

- **Type**: Real
- **Description**: Energy cutoff for charge density and potential, the unit is **Rydberg**. For norm-conserving pseudopotential you should stick to the default value, you can reduce it by a little but it will introduce noise especially on forces and stress. For ultrasoft pseudopotential a larger value than the default is often desirable (`ecutrho` = 8 to 12 times `ecutwfc`, typically). The use of gradient-corrected functional, especially in cells with vacuum, or for pseudopotential without non-linear core correction, usually requires an higher values of `ecutrho` to be accurately converged.
- **Default**: 4*ecutwfc

nx, ny, nz

- **Type**: Integer
- **Description**: If set to a positive number, then the three variables specify the numbers of FFT grid points in x, y, z directions, respectively. If set to 0, the number will be calculated from ecutrho.

  Note: You must specify all three dimensions for this setting to be used.
- **Default**: 0

ndx, ndy, ndz

- **Type**: Integer
- **Description**: If set to a positive number, then the three variables specify the numbers of FFT grid (for the dense part of charge density in ultrasoft pseudopotential) points in x, y, z directions, respectively. If set to 0, the number will be calculated from ecutwfc.

  Note: You must specify all three dimensions for this setting to be used.

  Note: These parameters must be used combined with *nx,ny,nz*. If *nx,ny,nz* are unset, ndx,ndy,ndz are used as *nx,ny,nz*.
- **Default**: 0

pw_seed

- **Type**: Integer
- **Description**: Only useful for plane wave basis only now. It is the random seed to initialize wave functions. Only positive integers are available.
- **Default**:0

pw_diag_thr

- **Type**: Real
- **Description**: Only used when you use `diago_type = cg` or `diago_type = david`. It indicates the threshold for the first electronic iteration, from the second iteration the pw_diag_thr will be updated automatically. **For nscf calculations with planewave basis set, pw_diag_thr should be <= 1e-3.**
- **Default**: 0.01

pw_diag_nmax

- **Type**: Integer
- **Description**: Only useful when you use `ks_solver = cg` or `ks_solver = dav`. It indicates the maximal iteration number for cg/david method.
- **Default**: 40

pw_diag_ndim

- **Type**: Integer
- **Description**: Only useful when you use `ks_solver = dav`. It indicates the maximal dimension for the Davidson method.
- **Default**: 4

erf_ecut

- **Type**: Real
- **Description**: Used in variable-cell molecular dynamics (or in stress calculation). See *erf_sigma* in detail.
- **Default**: 0.0
- **Unit**: Ry

fft_mode

- **Type**: Integer
- **Description**: Set the mode of FFTW.
  - 0: FFTW_ESTIMATE
  - 1: FFTW_MEASURE
  - 2: FFTW_PATIENT
  - 3: FFTW_EXHAUSTIVE
- **Default**: 0

erf_height

- **Type**: Real
- **Description**: Used in variable-cell molecular dynamics (or in stress calculation). See *erf_sigma* in detail.
- **Default**: 0.0
- **Unit**: Ry

erf_sigma

- **Type**: Real
- **Description**: In order to recover the accuracy of a constant energy cutoff calculation, the kinetic functional is modified, which is used in variable-cell molecular dynamics (or in stress calculation).

  erf_ecut is the value of the constant energy cutoff; *erf_height* and *erf_sigma* are the height and the width of the energy step for reciprocal vectors whose square modulus is greater than *erf_ecut*. In the kinetic energy, G^2 is replaced by G^2 + erf_height * (1 + erf ((G^2 - erf_ecut)/erf_sigma))

  See: M. Bernasconi et al., J. Phys. Chem. Solids **56**, 501 (1995), *doi:10.1016/0022-3697(94)00228-2*
- **Default**: 0.1
- **Unit**: Ry

back to top

13.1.4 Numerical atomic orbitals related variables

These variables are used to control the numerical atomic orbitals related parameters.

nb2d

- **Type**: Integer
- **Description**: In LCAO calculations, we arrange the total number of processors in an 2D array, so that we can partition the wavefunction matrix (number of bands*total size of atomic orbital basis) and distribute them in this 2D array. When the system is large, we group processors into sizes of nb2d, so that multiple processors take care of one row block (a group of atomic orbitals) in the wavefunction matrix. If set to 0, nb2d will be automatically set in the program according to the size of atomic orbital basis:
  - if size <= 500 : nb2d = 1
  - if 500 < size <= 1000 : nb2d = 32
  - if size > 1000 : nb2d = 64;
- **Default**: 0

lmaxmax

- **Type**: Integer
- **Description**: If not equals to 2, then the maximum l channels on LCAO is set to lmaxmax. If 2, then the number of l channels will be read from the LCAO data sets. Normally no input should be supplied for this variable so that it is kept as its default.
- **Default**: 2.

lcao_ecut

- **Type**: Real
- **Description**: Energy cutoff (in Ry) for two-center integrals in LCAO. The two-center integration table are obtained via a k space integral whose upper limit is about sqrt(`lcao_ecut`).
- **Default**: `ecutwfc`

lcao_dk

- **Type**: Real
- **Description**: k spacing (in Bohr^{-1}) for two-center integrals. The two-center integration table are obtained via a k space integral on a uniform grid with spacing `lcao_dk`.
- **Default**: 0.01

lcao_dr

- **Type**: Real
- **Description**: r spacing (in Bohr) of the integration table of two-center integrals.
- **Default**: 0.01

lcao_rmax

- **Type**: Real
- **Description**: Maximum distance (in Bohr) for the two-center integration table.
- **Default**: 30

search_radius

- **Type**: Real
- **Description**: Searching radius in finding the neighbouring atoms. By default the radius will be automatically determined by the cutoffs of orbitals and nonlocal beta projectors.
- **Default**: -1
- **Unit**: Bohr

search_pbc

- **Type**: Boolean
- **Description**: If True, periodic images will be included in searching for the neighbouring atoms. If False, periodic images will be ignored.
- **Default**: True

bx, by, bz

- **Type**: Integer
- **Description**: In the matrix operation of grid integral, bx/by/bz grids (in x, y, z directions) are treated as a whole as a matrix element. A different value will affect the calculation speed. The default is 0, which means abacus will automatically calculate these values.
- **Default**: 0

back to top

13.1.5 Electronic structure

These variables are used to control the electronic structure and geometry relaxation calculations.

basis_type

- **Type**: String
- **Description**: Choose the basis set.
  - **pw**: Using plane-wave basis set only.
  - **lcao**: Using localized atomic orbital sets.
  - **lcao_in_pw**: Expand the localized atomic set in plane-wave basis, non-self-consistent field calculation not tested.

- **Default**: pw

ks_solver

- **Type**: String
- **Description**: Choose the diagonalization methods for the Hamiltonian matrix expanded in a certain basis set.

  For plane-wave basis,

  - **cg**: cg method.
  - **bpcg**: bpcg method, which is a block-parallel Conjugate Gradient (CG) method, typically exhibits higher acceleration in a GPU environment.
  - **dav**: the Davidson algorithm.

  For atomic orbitals basis,

  - **genelpa**: This method should be used if you choose localized orbitals.
  - **scalapack_gvx**: Scalapack can also be used for localized orbitals.
  - **cusolver**: (Unavailable currently, it will be fixed in future versions) This method needs building with the cusolver component for lcao and at least one gpu is available.

  If you set ks_solver=`genelpa` for basis_type=`pw`, the program will be stopped with an error message:

```
genelpa can not be used with plane wave basis.
```

  Then the user has to correct the input file and restart the calculation.

- **Default**: cg (plane-wave basis), or genelpa (localized atomic orbital basis, if compiling option `USE_ELPA` has been set), scalapack_gvx, (localized atomic orbital basis, if compiling option `USE_ELPA` has not been set)

nbands

- **Type**: Integer
- **Description**: The number of Kohn-Sham orbitals to calculate. It is recommended to setup this value, especially when smearing techniques are utilized, more bands should be included.
- **Default**:
  - nspin=1: max(1.2*occupied_bands, occupied_bands + 10)
  - nspin=2: max(1.2*nelec_spin, nelec_spin + 10), in which nelec_spin = max(nelec_spin_up, nelec_spin_down)
  - nspin=4: max(1.2*nelec, nelec + 20)

nspin

- **Type**: Integer
- **Description**: The number of spin components of wave functions.
  - **1**: Spin degeneracy
  - **2**: Collinear spin polarized.
  - **4**: For the case of *noncollinear polarized*, nspin will be automatically set to 4 without being specified by the user.
- **Default**: 1

smearing_method

- **Type**: String
- **Description**: It indicates which occupation and smearing method is used in the calculation.
  - **fixed**: fixed occupations (available for non-coductors only)
  - **gauss** or **gaussian**: Gaussian smearing method.
  - **mp**: methfessel-paxton smearing method; recommended for metals.
  - **mp2**: 2-nd methfessel-paxton smearing method; recommended for metals.
  - **mv** or **cold**: marzari-vanderbilt smearing method.
  - **fd**: Fermi-Dirac smearing method: $f = 1/\{1 + \exp[(E - \mu)/kT]\}$ and smearing_sigma below is the temperature T (in Ry).
- **Default**: gauss

smearing_sigma

- **Type**: Real
- **Description**: Energy range for smearing.
- **Default**: 0.015
- **Unit**: Ry

smearing_sigma_temp

- **Type**: Real
- **Description**: Energy range for smearing, `smearing_sigma` = 1/2 * kB * `smearing_sigma_temp`.
- **Default**: 2 * `smearing_sigma` / kB.
- **Unit**: K

mixing_type

- **Type**: String
- **Description**: Charge mixing methods.
  - **plain**: Just simple mixing.
  - **pulay**: Standard Pulay method. P. Pulay Chemical Physics Letters, (1980)
  - **broyden**: Simplified modified Broyden method. D.D. Johnson Physical Review B (1988)

  In general, the convergence of the Broyden method is slightly faster than that of the Pulay method.
- **Default**: broyden

mixing_beta

- **Type**: Real
- **Description**: In general, the formula of charge mixing can be written as $\rho_{new} = \rho_{old} + \beta * \rho_{update}$, where ρ_{new} represents the new charge density after charge mixing, ρ_{old} represents the charge density in previous step, ρ_{update} is obtained through various mixing methods, and β is set by the parameter `mixing_beta`. A lower value of 'mixing_beta' results in less influence of ρ_{update} on ρ_{new}, making the self-consistent field (SCF) calculation more stable. However, it may require more steps to achieve convergence. We recommend the following options:
  - **0.8**: `nspin=1`
  - **0.4**: `nspin=2` and `nspin=4`
  - **0**: keep charge density unchanged, usually used for restarting with `init_chg=file` or testing.
  - **0.1 or less**: if convergence of SCF calculation is difficult to reach, please try `0 < mixing_beta < 0.1`.

  Note: For low-dimensional large systems, the setup of `mixing_beta=0.1`, `mixing_ndim=20`, and `mixing_gg0=1.0` usually works well.
- **Default**: 0.8 for `nspin=1`, 0.4 for `nspin=2` and `nspin=4`.

mixing_beta_mag

- **Type**: Real
- **Description**: Mixing parameter of magnetic density.
- **Default**: `4*mixing_beta`, but the maximum value is 1.6.

Note that `mixing_beta_mag` is not euqal to `mixing_beta` means that ρ_{up} and ρ_{down} mix independently from each other. This setting will fail for one case where the ρ_{up} and ρ_{down} of the ground state refers to different Kohn-Sham orbitals. For an atomic system, the ρ_{up} and ρ_{down} of the ground state refers to different Kohn-Sham orbitals. We all know Kohn-Sham orbitals are orthogonal to each other. So the mixture of ρ_{up} and ρ_{down} should be exactly independent, otherwise SCF cannot find the ground state forever. To sum up, please make sure `mixing_beta_mag` and `mixing_gg0_mag` exactly euqal to `mixing_beta` and `mixing_gg0` if you calculate an atomic system.

mixing_ndim

- **Type**: Integer
- **Description**: It indicates the mixing dimensions in Pulay or Broyden. Pulay and Broyden method use the density from previous mixing_ndim steps and do a charge mixing based on this density.

  For systems that are difficult to converge, one could try increasing the value of 'mixing_ndim' to enhance the stability of the self-consistent field (SCF) calculation.
- **Default**: 8

mixing_restart

- **Type**: double
- **Description**: If the density difference between input and output `drho` is smaller than `mixing_restart`, SCF will restart at next step which means SCF will restart by using output charge density from perivos iteration as input charge density directly, and start a new mixing. Notice that `mixing_restart` will only take effect once in one SCF.
- **Default**: 0

mixing_dmr

- **Type**: bool
- **Availability**: Only for `mixing_restart>=0.0`
- **Description**: At n-th iteration which is calculated by `drho<mixing_restart`, SCF will start a mixing for real-space density matrix by using the same coefficiences as the mixing of charge density.
- **Default**: false

mixing_gg0

- **Type**: Real
- **Description**: Whether to perfom Kerker scaling for charge density.
  - **>0**: The high frequency wave vectors will be suppressed by multiplying a scaling factor $\frac{k^2}{k^2+gg0^2}$. Setting `mixing_gg0 = 1.0` is normally a good starting point. Kerker preconditioner will be automatically turned off if `mixing_beta <= 0.1`.
  - **0**: No Kerker scaling is performed.

  For systems that are difficult to converge, particularly metallic systems, enabling Kerker scaling may aid in achieving convergence.
- **Default**: 1.0

mixing_gg0_mag

- **Type**: Real
- **Description**: Whether to perfom Kerker preconditioner of magnetic density. Note: we do not recommand to open Kerker preconditioner of magnetic density unless the system is too hard to converge.
- **Default**: 0.0

mixing_gg0_min

- **Type**: Real
- **Description**: the minimum kerker coefficient
- **Default**: 0.1

mixing_angle

- **Type**: Real
- **Availability**: Only relevant for non-colinear calculations `nspin=4`.
- **Description**: Normal broyden mixing can give the converged result for a given magnetic configuration. If one is not interested in the energies of a given magnetic configuration but wants to determine the ground state by relaxing the magnetic moments' directions, one cannot rely on the standard Broyden mixing algorithm. To enhance the ability to find correct magnetic configuration for non-colinear calculations, ABACUS implements a promising mixing method proposed by J. Phys. Soc. Jpn. 82 (2013) 114706. Here, `mixing_angle` is the angle mixing parameter. In fact, only `mixing_angle=1.0` is implemented currently.
  - **<=0**: Normal broyden mixing for m_x, m_y, m_z
  - **>0**: Angle mixing for the modulus $|m|$ with `mixing_angle=1.0`
- **Default**: -10.0

Note: In new angle mixing, you should set `mixing_beta_mag >> mixing_beta`. The setup of `mixing_beta=0.2`, `mixing_beta_mag=1.0` usually works well.

mixing_tau

- **Type**: Boolean
- **Availability**: Only relevant for meta-GGA calculations.
- **Description**: Whether to mix the kinetic energy density.
  - **True**: The kinetic energy density will also be mixed. It seems for general cases, SCF converges fine even without this mixing. However, if there is difficulty in converging SCF for meta-GGA, it might be helpful to turn this on.
  - **False**: The kinetic energy density will not be mixed.
- **Default**: False

mixing_dftu

- **Type**: Boolean
- **Availability**: Only relevant for DFT+U calculations.
- **Description**: Whether to mix the occupation matrices.
  - **True**: The occupation matrices will also be mixed by plain mixing. From experience this is not very helpful if the +U calculation does not converge.
  - **False**: The occupation matrices will not be mixed.
- **Default**: False

gamma_only

- **Type**: Integer
- **Availability**: Only used in localized orbitals set
- **Description**: Whether to use gamma_only algorithm.
  - **0**: more than one k-point is used and the ABACUS is slower compared to the gamma only algorithm.
  - **1**: ABACUS uses gamma only, the algorithm is faster and you don't need to specify the k-points file.

  Note: If gamma_only is set to 1, the KPT file will be overwritten. So make sure to turn off gamma_only for multi-k calculations.
- **Default**: 0

printe

- **Type**: Integer
- **Description**: Print out energy for each band for every printe step
- **Default**: 100

scf_nmax

- **Type**: Integer
- **Description**: This variable indicates the maximal iteration number for electronic iterations.
- **Default**: 100

scf_thr

- **Type**: Real
- **Description**: It's the threshold for electronic iteration. It represents the charge density error between two sequential densities from electronic iterations. Usually for local orbitals, usually 1e-6 may be accurate enough.
- **Default**: 1.0e-9 (plane-wave basis), or 1.0e-7 (localized atomic orbital basis).

scf_thr_type

- **Type**: Integer
- **Description**: Choose the calculation method of convergence criterion.
  - **1**: the criterion is defined as $\Delta\rho_G = \frac{1}{2}\iint \frac{\Delta\rho(r)\Delta\rho(r')}{|r-r'|}d^3rd^3r'$.
  - **2**: the criterion is defined as $\Delta\rho_R = \int |\Delta\rho(r)|d^3r$.

  Note: This parameter is still under testing and the default setting is usually sufficient.
- **Default**: 1 (plane-wave basis), or 2 (localized atomic orbital basis).

chg_extrap

- **Type**: String
- **Description**: Methods to do extrapolation of density when ABACUS is doing geometry relaxations or molecular dynamics.
  - **atomic**: atomic extrapolation.
  - **first-order**: first-order extrapolation.
  - **second-order**: second-order extrapolation.
- **Default**: first-order (geometry relaxations), second-order (molecular dynamics), else atomic

lspinorb

- **Type**: Boolean
- **Description**: Whether to consider spin-orbital coupling effect in the calculation.
  - **True**: Consider spin-orbital coupling effect, and `nspin` is also automatically set to 4.
  - **False**: Do not consider spin-orbital coupling effect.
- **Default**: False

noncolin

- **Type**: Boolean
- **Description**: Whether to allow non-collinear polarization, in which case the coupling between spin up and spin down will be taken into account.
  - **True**: Allow non-collinear polarization, and `nspin` is also automatically set to 4.
  - **False**: Do not allow non-collinear polarization.
- **Default**: False

soc_lambda

- **Type**: Real
- **Availability**: Relevant for soc calculations.
- **Description**: Sometimes, for some real materials, both scalar-relativistic and full-relativistic can not describe the exact spin-orbit coupling. Artificial modulation may help in such cases.

  `soc_lambda`, which has value range [0.0, 1.0] , is used for modulate SOC effect.

  In particular, `soc_lambda 0.0` refers to scalar-relativistic case and `soc_lambda 1.0` refers to full-relativistic case.
- **Default**: 1.0

back to top

13.1.6 Electronic structure (SDFT)

These variables are used to control the parameters of stochastic DFT (SDFT), mix stochastic-deterministic DFT (MDFT), or complete-basis Chebyshev method (CT). In the following text, stochastic DFT is used to refer to these three methods. We suggest using SDFT to calculate high-temperature systems and we only support *smearing_method* "fd". Both "scf" and "nscf" *calculation* are supported.

method_sto

- **Type**: Integer
- **Availability**: *esolver_type* = `sdft`
- **Description**: Different methods to do stochastic DFT
  - 1: Calculate $T_n(\hat{h})\chi$ twice, where $T_n(x)$ is the n-th order Chebyshev polynomial and $\hat{h} = \frac{\hat{H}-\bar{E}}{\Delta E}$ owning eigenvalues $\in (-1, 1)$. This method cost less memory but is slower.
  - 2: Calculate $T_n(\hat{h})\chi$ once but needs much more memory. This method is much faster. Besides, it calculates N_e with $\chi\sqrt{\hat{f}}\sqrt{\hat{f}}\chi$, which needs a smaller *nche_sto*. However, when the memory is not enough, only method 1 can be used.
  - other: use 2
- **Default**: 2

nbands_sto

- **Type**: Integer or string
- **Availability**: *esolver_type* = `sdft`
- **Description**: The number of stochastic orbitals
  - > 0: Perform stochastic DFT. Increasing the number of bands improves accuracy and reduces stochastic errors, which scale as $1/\sqrt{N_\chi}$; To perform mixed stochastic-deterministic DFT, you should set *nbands*, which represents the number of KS orbitals.
  - 0: Perform Kohn-Sham DFT.

  - all: All complete basis sets are used to replace stochastic orbitals with the Chebyshev method (CT), resulting in the same results as KSDFT without stochastic errors.

- **Default**: 256

nche_sto

- **Type**: Integer
- **Availability**: *esolver_type* = `sdft`
- **Description**: Chebyshev expansion orders for stochastic DFT.
- **Default**: 100

emin_sto

- **Type**: Real
- **Availability**: *esolver_type* = `sdft`
- **Description**: Trial energy to guess the lower bound of eigen energies of the Hamiltonian Operator $\hat{H}$.
- **Default**: 0.0
- **Unit**: Ry

emax_sto

- **Type**: Real
- **Availability**: *esolver_type* = `sdft`
- **Description**: Trial energy to guess the upper bound of eigen energies of the Hamiltonian Operator $\hat{H}$.
- **Default**: 0.0
- **Unit**: Ry

seed_sto

- **Type**: Integer
- **Availability**: *esolver_type* = `sdft`
- **Description**: The random seed to generate stochastic orbitals.
  - >= 0: Stochastic orbitals have the form of $\exp(i2\pi\theta(G))$, where θ is a uniform distribution in $(0, 1)$.
  - 0: the seed is decided by time(NULL).
  - <= -1: Stochastic orbitals have the form of ± 1 with equal probability.
  - -1: the seed is decided by time(NULL).
- **Default**: 0

initsto_ecut

- **Type**: Real
- **Availability**: *esolver_type* = `sdft`
- **Description**: Stochastic wave functions are initialized in a large box generated by "4*`initsto_ecut`". `initsto_ecut` should be larger than *ecutwfc*. In this method, SDFT results are the same when using different cores. Besides, coefficients of the same G are the same when ecutwfc is rising to initsto_ecut. If it is smaller than *ecutwfc*, it will be turned off.
- **Default**: 0.0
- **Unit**: Ry

initsto_freq

- **Type**: Integer
- **Availability**: *esolver_type* = `sdft`
- **Description**: Frequency (once each initsto_freq steps) to generate new stochastic orbitals when running md.
  - positive integer: Update stochastic orbitals
  - 0: Never change stochastic orbitals.
- **Default**: 0

npart_sto

- **Type**: Integer
- **Availability**: *method_sto* = `2` and *out_dos* = `True` or *cal_cond* = `True`
- **Description**: Make memory cost to 1/npart_sto times of the previous one when running the post process of SDFT like DOS or conductivities.
- **Default**: 1

back to top

13.1.7 Geometry relaxation

These variables are used to control the geometry relaxation.

relax_method

- **Type**: String
- **Description**: The methods to do geometry optimization.
  - cg: using the conjugate gradient (CG) algorithm. Note that there are two implementations of the conjugate gradient (CG) method, see *relax_new*.
  - bfgs: using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.
  - cg_bfgs: using the CG method for the initial steps, and switching to BFGS method when the force convergence is smaller than *relax_cg_thr*.

  - sd: using the steepest descent (SD) algorithm.
  - fire: the Fast Inertial Relaxation Engine method (FIRE), a kind of molecular-dynamics-based relaxation algorithm, is implemented in the molecular dynamics (MD) module. The algorithm can be used by setting *calculation* to `md` and *md_type* to `fire`. Also ionic velocities should be set in this case. See *fire* for more details.

- **Default**: cg

relax_new

- **Type**: Boolean
- **Description**: At around the end of 2022 we made a new implementation of the Conjugate Gradient (CG) method for `relax` and `cell-relax` calculations. But the old implementation was also kept.
  - True: use the new implementation of CG method for `relax` and `cell-relax` calculations.
  - False: use the old implementation of CG method for `relax` and `cell-relax` calculations.
- **Default**: True

relax_scale_force

- **Type**: Real
- **Availability**: only used when `relax_new` set to `True`
- **Description**: The parameter controls the size of the first conjugate gradient step. A smaller value means the first step along a new CG direction is smaller. This might be helpful for large systems, where it is safer to take a smaller initial step to prevent the collapse of the whole configuration.
- **Default**: 0.5

relax_nmax

- **Type**: Integer
- **Description**: The maximal number of ionic iteration steps, the minimum value is 1.
- **Default**: 1

relax_cg_thr

- **Type**: Real
- **Description**: When move-method is set to `cg_bfgs`, a mixed algorithm of conjugate gradient (CG) method and Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is used. The ions first move according to CG method, then switched to BFGS method when the maximum of force on atoms is reduced below the CG force threshold, which is set by this parameter.
- **Default**: 0.5
- **Unit**: eV/Angstrom

cal_force

- **Type**: Boolean
- **Description**:
  - **True** calculate the force at the end of the electronic iteration
  - **False** no force calculation at the end of the electronic iteration
- **Default**: False if `calculation` is set to `scf`, True if `calculation` is set to `cell-relax`, `relax`, or `md`.

force_thr

- **Type**: Real
- **Description**: Threshold of the force convergence in Ry/Bohr. The threshold is compared with the largest force among all of the atoms. The recommended value for using atomic orbitals is 0.04 eV/Angstrom (0.0016 Ry/Bohr). The parameter is equivalent to *force_thr_ev* except for the unit. You may choose either you like.
- **Default**: 0.001
- **Unit**: Ry/Bohr (25.7112 eV/Angstrom)

force_thr_ev

- **Type**: Real
- **Description**: Threshold of the force convergence in eV/Angstrom. The threshold is compared with the largest force among all of the atoms. The recommended value for using atomic orbitals is 0.04 eV/Angstrom (0.0016 Ry/Bohr). The parameter is equivalent to *force_thr* except for the unit. You may choose either you like.
- **Default**: 0.0257112
- **Unit**: eV/Angstrom (0.03889 Ry/Bohr)

force_thr_ev2

- **Type**: Real
- **Description**: The calculated force will be set to 0 when it is smaller than the parameter `force_thr_ev2`.
- **Default**: 0.0
- **Unit**: eV/Angstrom

relax_bfgs_w1

- **Type**: Real
- **Description**: This variable controls the Wolfe condition for Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm used in geometry relaxation. You can look into the paper Phys.Chem.Chem.Phys.,2000,2,2177 for more information.
- **Default**: 0.01

relax_bfgs_w2

- **Type**: Real
- **Description**: This variable controls the Wolfe condition for Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm used in geometry relaxation. You can look into the paper Phys.Chem.Chem.Phys.,2000,2,2177 for more information.
- **Default**: 0.5

relax_bfgs_rmax

- **Type**: Real
- **Description**: This variable is for geometry optimization. It stands for the maximal movement of all the atoms. The sum of the movements from all atoms can be increased during the optimization steps. However, it can not be larger than `relax_bfgs_rmax`
- **Unit**: Bohr
- **Default**: 0.8

relax_bfgs_rmin

- **Type**: Real
- **Description**: This variable is for geometry optimization. It indicates the minimal movement of all the atoms. When the movement of all the atoms is smaller than relax_bfgs_rmin Bohr, and the force convergence is still not achieved, the calculation will break down.
- **Default**: 1e-5
- **Unit**: Bohr

relax_bfgs_init

- **Type**: Real
- **Description**: This variable is for geometry optimization. It stands for the sum of initial movements of all of the atoms.
- **Default**: 0.5
- **Unit**: Bohr

cal_stress

- **Type**: Boolean
- **Description**:
  - **True**: calculate the stress at the end of the electronic iteration
  - **False**: no calculation of the stress at the end of the electronic iteration
- **Default**: True if `calculation` is `cell-relax`, False otherwise.

stress_thr

- **Type**: Real
- **Description**: The threshold of the stress convergence. The threshold is compared with the largest component of the stress tensor.
- **Default**: 0.5
- **Unit**: kbar

press1, press2, press3

- **Type**: Real
- **Description**: The external pressures along three axes. Positive input value is taken as compressive stress.
- **Default**: 0
- **Unit**: kbar

fixed_axes

- **Type**: String
- **Availability**: only used when `calculation` set to `cell-relax`
- **Description**: Axes that are fixed during cell relaxation. Possible choices are:
  - **None**: default; all of the axes can relax
  - **volume**: relaxation with fixed volume
  - **shape**: fix shape but change volume (i.e. only lattice constant changes)
  - **a**: fix a axis during relaxation
  - **b**: fix b axis during relaxation
  - **c**: fix c axis during relaxation
  - **ab**: fix both a and b axes during relaxation
  - **ac**: fix both a and c axes during relaxation
  - **bc**: fix both b and c axes during relaxation

  Note : fixed_axes = "shape" and "volume" are only available for *relax_new* = True
- **Default**: None

fixed_ibrav

- **Type**: Boolean
- **Availability**: Must be used along with *relax_new* set to True, and a specific *latname* must be provided
- **Description**:
  - **True**: the lattice type will be preserved during relaxation
  - **False**: No restrictions are exerted during relaxation in terms of lattice type

Note: it is possible to use `fixed_ibrav` with `fixed_axes`, but please make sure you know what you are doing. For example, if we are doing relaxation of a simple cubic lattice (`latname` = "sc"), and we use `fixed_ibrav` along with `fixed_axes` = "volume", then the cell is never allowed to move and as a result, the relaxation never converges.

- **Default**: False

fixed_atoms

- **Type**: Boolean
- **Description**:
  - **True**: The direct coordinates of atoms will be preserved during variable-cell relaxation.
  - **False**: No restrictions are exerted on positions of all atoms. However, users can still fix certain components of certain atoms by using the `m` keyword in `STRU` file. For the latter option, check the end of this *instruction*.
- **Default**: False

cell_factor

- **Type**: Real
- **Description**: Used in the construction of the pseudopotential tables. It should exceed the maximum linear contraction of the cell during a simulation.
- **Default**: 1.2

back to top

13.1.8 Variables related to output information

These variables are used to control the output of properties.

out_mul

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis
- **Description**: Whether to print the Mulliken population analysis result into `OUT.${suffix}/mulliken.txt`. In molecular dynamics calculations, the output frequency is controlled by *out_interval*.
- **Default**: False

out_freq_elec

- **Type**: Integer
- **Description**: The output frequency of the charge density (controlled by *out_chg*), wavefunction (controlled by *out_wfc_pw* or *out_wfc_r*), and density matrix of localized orbitals (controlled by *out_dm*).
  - >0: Output them every `out_freq_elec` iteration numbers in electronic iterations.
  - 0: Output them when the electronic iteration is converged or reaches the maximal iteration number.
- **Default**: 0

out_chg

- **Type**: Boolean
- **Description**: Whether to output the charge density (in Bohr^-3) on real space grids into the density files in the folder `OUT.${suffix}`. The files are named as:
  - npsin = 1: SPIN1_CHG.cube;
  - npsin = 2: SPIN1_CHG.cube, and SPIN2_CHG.cube;
  - npsin = 4: SPIN1_CHG.cube, SPIN2_CHG.cube, SPIN3_CHG.cube, and SPIN4_CHG.cube.

  The circle order of the charge density on real space grids is: x is the outer loop, then y and finally z (z is moving fastest).

  If EXX(exact exchange) is calculated, (i.e. dft_fuctional==*hse/hf/pbe0/scan0/opt_orb* or rpa==*True*), the Hexx® files will be output in the folder `OUT.${suffix}` too, which can be read in NSCF calculation.
- **Default**: False

out_pot

- **Type**: Integer
- **Description**:
  - 1: Output the **total local potential** (i.e., local pseudopotential + Hartree potential + XC potential + external electric field (if exists) + dipole correction potential (if exists) + …) on real space grids (in Ry) into files in the folder `OUT.${suffix}`. The files are named as:
    - npsin = 1: SPIN1_POT.cube;
    - npsin = 2: SPIN1_POT.cube, and SPIN2_POT.cube;
    - npsin = 4: SPIN1_POT.cube, SPIN2_POT.cube, SPIN3_POT.cube, and SPIN4_POT.cube.
  - 2: Output the **electrostatic potential** on real space grids into `OUT.${suffix}/ElecStaticPot.cube`. The Python script named `tools/average_pot/aveElecStatPot.py` can be used to calculate the average electrostatic potential along the z-axis and outputs it into ElecStaticPot_AVE.

    Please note that the total local potential refers to the local component of the self-consistent potential, excluding the non-local pseudopotential. The distinction between the local potential and the electrostatic potential is as follows: local potential = electrostatic potential + XC potential.
- **Default**: 0

out_dm

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (gamma-only algorithm)
- **Description**: Whether to output the density matrix of localized orbitals into files in the folder `OUT.${suffix}`. The files are named as:
  - npsin = 1: SPIN1_DM;
  - npsin = 2: SPIN1_DM, and SPIN2_DM.
- **Default**: False

out_dm1

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (multi-k points)
- **Description**: Whether to output the density matrix of localized orbitals into files in the folder `OUT.${suffix}`. The density matrices are written in the format of sparse matrices, as mentioned in *out_mat_hs2*. The files are named as:
  - npsin = 1: data-DMR-sparse_SPIN0.csr;
  - npsin = 2: data-DMR-sparse_SPIN0.csr, and data-DMR-sparse_SPIN1.csr.
- **Default**: False

out_wfc_pw

- **Type**: Integer
- **Availability**: Plane wave basis or get_wf calculation in numerical atomic orbital basis
- **Description**:
  - 1: Output the coefficients of wave functions into text files named `OUT.${suffix}/WAVEFUNC${K}.txt`, where ${K} is the index of k points.
  - 2: results are stored in binary files named `OUT.${suffix}/WAVEFUNC${K}.dat`.
- **Default**: 0

out_wfc_r

- **Type**: Boolean
- **Availability**: Plane wave basis or get_wf calculation in numerical atomic orbital basis
- **Description**: Whether to output real-space wave functions into `OUT.suffix/wfc_realspace/wfc_realspace_${K}_${B}`, where `${K}` is the index of k points, `${B}` is the index of bands.
- **Default**: False

out_wfc_lcao

- **Type**: Integer
- **Availability**: Numerical atomic orbital basis
- **Description**: Whether to output the wavefunction coefficients into files in the folder `OUT.${suffix}`. The files are named as:
  - 0: no output
  - 1: (txt format)
    - gamma-only: `LOWF_GAMMA_S1.txt`;
    - non-gamma-only: `LOWF_K_${k}.txt`, where `${k}` is the index of k points.
  - 2: (binary format)
    - gamma-only: `LOWF_GAMMA_S1.dat`;
    - non-gamma-only: `LOWF_K_${k}.dat`, where `${k}` is the index of k points.

  The corresponding sequence of the orbitals can be seen in *Basis Set*.

  Also controled by *out_interval* and *out_app_flag*.
- **Default**: Flase

out_dos

- **Type**: Boolean
- **Description**: Whether to output the density of states (DOS). For more information, refer to the *dos.md*.
- **Default**: False

out_band

- **Type**: Boolean Integer(optional)
- **Description**: Whether to output the band structure (in eV), optionally output precision can be set by a second parameter, default is 8. For more information, refer to the *band.md*
- **Default**: False

out_proj_band

- **Type**: Boolean
- **Description**: Whether to output the projected band structure. For more information, refer to the *band.md*
- **Default**: False

out_stru

- **Type**: Boolean
- **Description**: Whether to output structure files per ionic step in geometry relaxation calculations into `OUT.${suffix}/STRU_ION${istep}_D`, where `${istep}` is the ionic step.
- **Default**: False

out_bandgap

- **Type**: Boolean
- **Description**: Whether to print the bandgap per electronic iteration into `OUT.${suffix}/running_${calculation}.log`. The value of bandgaps can be obtained by searching for the keyword:
  - *nupdown* > 0: `E_bandgap_up` and `E_bandgap_dw`
  - *nupdown* = 0: `E_bandgap`
- **Default**: False

out_level

- **Type**: String
- **Description**: Control the output level of information in `OUT.${suffix}/running_${calculation}.log`.
  - ie: electronic iteration level, which prints useful information for electronic iterations;
  - i: geometry relaxation level, which prints some information for geometry relaxations additionally;
  - m: molecular dynamics level, which does not print some information for simplicity.
- **Default**: ie

out_alllog

- **Type**: Boolean
- **Description**: Whether to print information into individual logs from all ranks in an MPI run.
  - True: Information from each rank will be written into individual files named `OUT.${suffix}/running_${calculation}_${rank+1}.log`.
  - False: Information will only be written from rank 0 into a file named `OUT.${suffix}/running_${calculation}.log`.
- **Default**: False

out_mat_hs

- **Type**: Boolean Integer(optional)
- **Availability**: Numerical atomic orbital basis
- **Description**: Whether to print the upper triangular part of the Hamiltonian matrices (in Ry) and overlap matrices for each k point into files in the directory `OUT.${suffix}`. The second number controls precision. For more information, please refer to *hs_matrix.md*. Also controled by *out_interval* and *out_app_flag*.
- **Default**: False 8

out_mat_r

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (not gamma-only algorithm)
- **Description**: Whether to print the matrix representation of the position matrix (in Bohr) into a file named `data-rR-tr` in the directory `OUT.${suffix}`. For more information, please refer to position_matrix.md.
- **Default**: False

out_mat_hs2

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (not gamma-only algorithm)
- **Description**: Whether to print files containing the Hamiltonian matrix $H(R)$ (in Ry) and overlap matrix $S(R)$ into files in the directory `OUT.${suffix}`. For more information, please refer to *hs_matrix.md*.
- **Default**: False

out_mat_t

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (not gamma-only algorithm)
- **Description**: For LCAO calculations, if out_mat_t is set to 1, ABACUS will generate files containing the kinetic energy matrix $T(R)$ (in Ry). The format will be the same as the Hamiltonian matrix $H(R)$ and overlap matrix $S(R)$ as mentioned in *out_mat_hs2*. The name of the files will be `data-TR-sparse_SPIN0.csr` and so on. Also controled by *out_interval* and *out_app_flag*.
- **Default**: False

out_mat_dh

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (not gamma-only algorithm)
- **Description**: Whether to print files containing the derivatives of the Hamiltonian matrix (in Ry/Bohr). The format will be the same as the Hamiltonian matrix $H(R)$ and overlap matrix $S(R)$ as mentioned in *out_mat_hs2*. The name of the files will be `data-dHRx-sparse_SPIN0.csr` and so on. Also controled by *out_interval* and *out_app_flag*.

- **Default**: False

out_mat_xc

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis
- **Description**: Whether to print the upper triangular part of the exchange-correlation matrices in **Kohn-Sham orbital representation** (unit: Ry): $\psi_i|V_{\text{xc}}^{\text{(semi-)local}} + V_{\text{exx}} + V_{\text{DFTU}}|\psi_j$ for each k point into files in the directory `OUT.${suffix}`, which is useful for the subsequent GW calculation. (Note that currently DeePKS term is not included.) The files are named `k-$k-Vxc`, the meaning of `$k`corresponding to k point and spin is same as *hs_matrix.md*.
- **Default**: False

out_app_flag

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis (not gamma-only algorithm)
- **Description**: Whether to output $r(R)$, $H(R)$, $S(R)$, $T(R)$, $dH(R)$, $H(k)$, $S(k)$ and $wfc(k)$ matrices in an append manner during molecular dynamics calculations. Check input parameters *out_mat_r*, *out_mat_hs2*, *out_mat_t*, *out_mat_dh*, *out_mat_hs* and *out_wfc_lcao* for more information.
- **Default**: true

out_ndigits

- **Type**: Integar
- **Availability**: `out_mat_hs 1` case presently.
- **Description**: Controls the length of decimal part of output data, such as charge density, Hamiltonian matrix, Overlap matrix and so on.
- **Default**: 8

out_interval

- **Type**: Integer
- **Availability**: Numerical atomic orbital basis
- **Description**: Control the interval for printing Mulliken population analysis, $r(R)$, $H(R)$, $S(R)$, $T(R)$, $dH(R)$, $H(k)$, $S(k)$ and $wfc(k)$ matrices during molecular dynamics calculations. Check input parameters *out_mul*, *out_mat_r*, *out_mat_hs2*, *out_mat_t*, *out_mat_dh*, *out_mat_hs* and *out_wfc_lcao* for more information, respectively.
- **Default**: 1

out_element_info

- **Type**: Boolean
- **Description**: Whether to print element information into files in the directory `OUT.${suffix}/${element_label}`, including pseudopotential and orbital information of the element (in atomic Ryberg units).
- **Default**: False

restart_save

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis
- **Description**: Whether to save charge density files per ionic step, which are used to restart calculations. According to the value of *read_file_dir*:
  - auto: These files are saved in folder `OUT.${suffix}/restart/`;
  - other: These files are saved in folder `${read_file_dir}/restart/`.

  If EXX(exact exchange) is calculated (i.e. dft_fuctional==*hse/hf/pbe0/scan0/opt_orb* or rpa==*True*), the Hexx(k) files for each k-point will also be saved in the above folder, which can be read in EXX calculation with restart_load==*True*.
- **Default**: False

restart_load

- **Type**: Boolean
- **Availability**: Numerical atomic orbital basis
- **Description**: If *restart_save* is set to true and an electronic iteration is finished, calculations can be restarted from the charge density file, which are saved in the former calculation. Please ensure *read_file_dir* is correct, and the charge density file exist.

  If EXX(exact exchange) is calculated (i.e. dft_fuctional==*hse/hf/pbe0/scan0/opt_orb* or rpa==*True*), the Hexx(k) files in the same folder for each k-point will also be read.
- **Default**: False

rpa

- **Type**: Boolean
- **Description**: Generate output files used in rpa calculations.
- **Default**: False

nbands_istate

- **Type**: Integer
- **Availability**: Only for LCAO, used when `calculation = get_wf` or `calculation = get_pchg`.
- **Description**: The number of bands around the Fermi level you would like to calculate. `get_wf` means to calculate the envelope functions of wave functions $\Psi_i = \Sigma_\mu C_{i\mu}\Phi_\mu$, where Ψ_i is the ith wave function with the band index i and Φ_μ is the localized atomic orbital set. `get_pchg` means to calculate the density of each wave function $|\Psi_i|^2$. Specifically, suppose we have highest occupied bands at 100th wave functions. And if you set this variable to 5, it will print five wave functions from 96th to 105th. But before all this can be carried out, the wave functions coefficients should be first calculated and written into a file by setting the flag `out_wfc_lcao = 1`.
- **Default**: 5

bands_to_print

- **Type**: String
- **Availability**: For both PW and LCAO. When `basis_type = lcao`, only used when `calculation = get_pchg`.
- **Description**: Specifies the bands to calculate the charge density for, using a space-separated string of 0s and 1s, providing a more flexible selection compared to `nbands_istate`. Each digit in the string corresponds to a band, starting from the first band. A `1` indicates that the charge density should be calculated for that band, while a `0` means the band will be ignored. The parameter allows a compact and flexible notation (similar to *`ocp_set`*), for example the syntax `1 4*0 5*1 0` is used to denote the selection of bands: `1` means calculate for the first band, `4*0` skips the next four bands, `5*1` means calculate for the following five bands, and the final `0` skips the next band. It's essential that the total count of bands does not exceed the total number of bands (`nbands`); otherwise, it results in an error, and the process exits. The input string must contain only numbers and the asterisk (`*`) for repetition, ensuring correct format and intention of band selection.
- **Default**: none

back to top

13.1.9 Density of states

These variables are used to control the calculation of DOS. Detailed introduction

dos_edelta_ev

- **Type**: Real
- **Description**: the step size in writing Density of States (DOS)
- **Default**: 0.01
- **Unit**: eV

dos_sigma

- **Type**: Real
- **Description**: the width of the Gaussian factor when obtaining smeared Density of States (DOS)
- **Default**: 0.07
- **Unit**: eV

dos_scale

- **Type**: Real
- **Description**: Defines the energy range of DOS output as (emax-emin)*(1+dos_scale), centered at (emax+emin)/2. This parameter will be used when dos_emin and dos_emax are not set.
- **Default**: 0.01
- **Unit**: eV

dos_emin_ev

- **Type**: Real
- **Description**: the minimal range for Density of States (DOS)
  - If set, "dos_scale" will be ignored.
- **Default**: Minimal eigenenergy of $\hat{H}$
- **Unit**: eV

dos_emax_ev

- **Type**: Real
- **Description**: the maximal range for Density of States (DOS)
  - If set, "dos_scale" will be ignored.
- **Default**: Maximal eigenenergy of $\hat{H}$
- **Unit**: eV

dos_nche

- **Type**: Integer The order of Chebyshev expansions when using Stochastic Density Functional Theory (SDFT) to calculate DOS.
- **Default**: 100

back to top

13.1.10 NAOs

These variables are used to control the generation of numerical atomic orbitals (NAOs), whose radial parts are linear combinations of spherical Bessel functions with a node (i.e., evaluate to zero) at the cutoff radius. In plane-wave-based calculations, necessary information will be printed into `OUT.${suffix}/orb_matrix.${i}.dat`, which serves as an input file for the generation of NAOs. Please check SIAB package for more information.

bessel_nao_ecut

- **Type**: Real
- **Description**: "Energy cutoff" (in Ry) of spherical Bessel functions. The number of spherical Bessel functions that constitute the radial parts of NAOs is determined by sqrt(`bessel_nao_ecut`)×`bessel_nao_rcut`/π.
- **Default**: `ecutwfc`

bessel_nao_tolerence

- **Type**: Real
- **Description**: tolerance when searching for the zeros of spherical Bessel functions.
- **Default**: 1.0e-12

bessel_nao_rcut

- **Type**: Real
- **Description**: Cutoff radius (in Bohr) and the common node of spherical Bessel functions used to construct the NAOs.
- **Default**: 6.0

bessel_nao_smooth

- **Type**: Boolean
- **Description**: if True, NAOs will be smoothed near the cutoff radius by $1 - \exp\left(-\frac{(r-r_{cut})^2}{2\sigma^2}\right)$. See `bessel_nao_rcut` for r_{cut} and `bessel_nao_sigma` for σ.
- **Default**: True

bessel_nao_sigma

- **Type**: Real
- **Description**: Smoothing range (in Bohr). See also `bessel_nao_smooth`.
- **Default**: 0.1

back to top

13.1.11 DeePKS

These variables are used to control the usage of DeePKS method (a comprehensive data-driven approach to improve the accuracy of DFT). Warning: this function is not robust enough for the current version. Please try the following variables at your own risk:

deepks_out_labels

- **Type**: Boolean
- **Availability**: numerical atomic orbital basis
- **Description**: print energy and force labels and descriptors for DeePKS training
- **Note**: In `LCAO` calculation, the path of a numerical descriptor (an `orb` file) is needed to be specified under the `NUMERICAL_DESCRIPTOR` tag in the `STRU` file. For example:

```
NUMERICAL_ORBITAL
H_gga_8au_60Ry_2s1p.orb
O_gga_7au_60Ry_2s2p1d.orb

NUMERICAL_DESCRIPTOR
jle.orb
```

- **Default**: False

deepks_scf

- **Type**: Boolean
- **Availability**: numerical atomic orbital basis
- **Description**: perform self-consistent field iteration in DeePKS method
- **Note**: A trained, traced model file is needed.
- **Default**: False

deepks_model

- **Type**: String
- **Availability**: numerical atomic orbital basis and `deepks_scf` is true
- **Description**: the path of the trained, traced neural network model file generated by deepks-kit
- **Default**: None

bessel_descriptor_lmax

- **Type**: Integer
- **Availability**: `gen_bessel` calculation
- **Description**: the maximum angular momentum of the Bessel functions generated as the projectors in DeePKS
- **NOte**: To generate such projectors, set calculation type to `gen_bessel` in ABACUS. See also *calculation*.
- **Default**: 2

bessel_descriptor_ecut

- **Type**: Real
- **Availability**: `gen_bessel` calculation
- **Description**: energy cutoff of Bessel functions
- **Default**: same as ecutwfc
- **Unit**: Ry

bessel_descriptor_tolerence

- **Type**: Real
- **Availability**: `gen_bessel` calculation
- **Description**: tolerance for searching the zeros of Bessel functions
- **Default**: 1.0e-12

bessel_descriptor_rcut

- **Type**: Real
- **Availability**: `gen_bessel` calculation
- **Description**: cutoff radius of Bessel functions
- **Default**: 6.0
- **Unit**: Bohr

bessel_descriptor_smooth

- **Type**: Boolean
- **Availability**: `gen_bessel` calculation
- **Description**: smooth the Bessel functions at radius cutoff
- **Default**: False

bessel_descriptor_sigma

- **Type**: Real
- **Availability**: `gen_bessel` calculation
- **Description**: smooth parameter at the cutoff radius of projectors
- **Default**: 0.1
- **Unit**: Bohr

deepks_bandgap

- **Type**: Boolean
- **Availability**: numerical atomic orbital basis and `deepks_scf` is true
- **Description**: include bandgap label for DeePKS training
- **Default**: False

deepks_out_unittest

- **Type**: Boolean
- **Description**: generate files for constructing DeePKS unit test
- **Note**: Not relevant when running actual calculations. When set to 1, ABACUS needs to be run with only 1 process.
- **Default**: False

back to top

13.1.12 OFDFT: orbital free density functional theory

of_kinetic

- **Type**: String
- **Availability**: OFDFT
- **Description**: The type of KEDF (kinetic energy density functional).
  - **wt**: Wang-Teter.
  - **tf**: Thomas-Fermi.
  - **vw**: von Weizsäcker.
  - **tf+**: TFλvW, the parameter λ can be set by `of_vw_weight`.
  - **lkt**: Luo-Karasiev-Trickey.
- **Default**: wt

of_method

- **Type**: String
- **Availability**: OFDFT
- **Description**: The optimization method used in OFDFT.
  - **cg1**: Polak-Ribiere. Standard CG algorithm.
  - **cg2**: Hager-Zhang (generally faster than cg1).
  - **tn**: Truncated Newton algorithm.
- **Default**:tn

of_conv

- **Type**: String
- **Availability**: OFDFT
- **Description**: Criterion used to check the convergence of OFDFT.
  - **energy**: Ttotal energy changes less than `of_tole`.
  - **potential**: The norm of potential is less than `of_tolp`.
  - **both**: Both energy and potential must satisfy the convergence criterion.
- **Default**: energy

of_tole

- **Type**: Real
- **Availability**: OFDFT
- **Description**: Tolerance of the energy change for determining the convergence.
- **Default**: 2e-6
- **Unit**: Ry

of_tolp

- **Type**: Real
- **Availability**: OFDFT
- **Description**: Tolerance of potential for determining the convergence.
- **Default**: 1e-5
- **Unit**: Ry

of_tf_weight

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=tf, tf+, wt`
- **Description**: Weight of TF KEDF (kinetic energy density functional).
- **Default**: 1.0

of_vw_weight

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=vw, tf+, wt, lkt`
- **Description**: Weight of vW KEDF (kinetic energy density functional).
- **Default**: 1.0

of_wt_alpha

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=wt`
- **Description**: Parameter alpha of WT KEDF (kinetic energy density functional).
- **Default**: $5/6$

of_wt_beta

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=wt`
- **Description**: Parameter beta of WT KEDF (kinetic energy density functional).
- **Default**: $5/6$

of_wt_rho0

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=wt`
- **Description**: The average density of system.
- **Default**: 0.0
- **Unit**: Bohr^-3

of_hold_rho0

- **Type**: Boolean
- **Availability**: OFDFT with `of_kinetic=wt`
- **Description**: Whether to fix the average density rho0.
  - **True**: rho0 will be fixed even if the volume of system has changed, it will be set to True automatically if `of_wt_rho0` is not zero.
  - **False**: rho0 will change if volume of system has changed.
- **Default**: False

of_lkt_a

- **Type**: Real
- **Availability**: OFDFT with `of_kinetic=lkt`
- **Description**: Parameter a of LKT KEDF (kinetic energy density functional).
- **Default**: 1.3

of_read_kernel

- **Type**: Boolean
- **Availability**: OFDFT with `of_kinetic=wt`
- **Description**: Whether to read in the kernel file.
  - **True**: The kernel of WT KEDF (kinetic energy density functional) will be filled from the file specified by `of_kernel_file`.
  - **False**: The kernel of WT KEDF (kinetic energy density functional) will be filled from formula.
- **Default**: False

of_kernel_file

- **Type**: String
- **Availability**: OFDFT with `of_read_kernel=True`
- **Description**: The name of WT kernel file.
- **Default**: WTkernel.txt

of_full_pw

- **Type**: Boolean
- **Availability**: OFDFT
- **Description**: Whether to use full planewaves.
  - **True**: Ecut will be ignored while collecting planewaves, so that all planewaves will be used in FFT.
  - **False**: Only use the planewaves inside ecut, the same as KSDFT.
- **Default**: True

of_full_pw_dim

- **Type**: Integer
- **Availability**: OFDFT with `of_full_pw = True`
- **Description**: Specify the parity of FFT dimensions.
  - **0**: either odd or even.
  - **1**: odd only.
  - **2**: even only.

  Note: Even dimensions may cause slight errors in FFT. It should be ignorable in ofdft calculation, but it may make Cardinal B-spline interpolation unstable, so please set `of_full_pw_dim = 1` if `nbspline != -1`.
- **Default**: 0

back to top

13.1.13 Electric field and dipole correction

These variables are relevant to electric field and dipole correction

efield_flag

- **Type**: Boolean
- **Description**: added the electric field.
  - True: A saw-like potential simulating an electric field is added to the bare ionic potential.
  - False: Not added the electric field.
- **Default**: False

dip_cor_flag

- **Type**: Boolean
- **Availability**: with dip_cor_flag = True and efield_flag = True.
- **Description**: Added a dipole correction to the bare ionic potential.
  - True：A dipole correction is also added to the bare ionic potential.
  - False: A dipole correction is not added to the bare ionic potential.

  Note: If you want no electric field, parameter efield_amp should be zero. Must be used ONLY in a slab geometry for surface alculations, with the discontinuity FALLING IN THE EMPTY SPACE.
- **Default**: False

efield_dir

- **Type**: Integer
- **Availability**: with efield_flag = True.
- **Description**: The direction of the electric field or dipole correction is parallel to the reciprocal lattice vector, so the potential is constant in planes defined by FFT grid points, efield_dir can set to 0, 1 or 2.
  - 0: parallel to $b_1 = \frac{2\pi(a_2 \times a_3)}{a_1 \cdot (a_2 \times a_3)}$
  - 1: parallel to $b_2 = \frac{2\pi(a_3 \times a_1)}{a_1 \cdot (a_2 \times a_3)}$
  - 2: parallel to $b_3 = \frac{2\pi(a_1 \times a_2)}{a_1 \cdot (a_2 \times a_3)}$
- **Default**: 2

efield_pos_max

- **Type**: Real
- **Availability**: with efield_flag = True.
- **Description**: Position of the maximum of the saw-like potential along crystal axis efield_dir, within the unit cell, 0 <= efield_pos_max < 1.
- **Default**: Autoset to `center of vacuum - width of vacuum / 20`

efield_pos_dec

- **Type**: Real
- **Availability**: with efield_flag = True.
- **Description**: Zone in the unit cell where the saw-like potential decreases, 0 < efield_pos_dec < 1.
- **Default**: Autoset to `width of vacuum / 10`

efield_amp

- **Type**: Real
- **Availability**: with efield_flag = True.
- **Description**: Amplitude of the electric field. The saw-like potential increases with slope efield_amp in the region from efield_pos_max+efield_pos_dec-1) to (efield_pos_max), then decreases until (efield_pos_max+efield_pos_dec), in units of the crystal vector efield_dir.

  Note: The change of slope of this potential must be located in the empty region, or else unphysical forces will result.
- **Default**: 0.0
- **Unit**: a.u., 1 a.u. = 51.4220632*10^10 V/m.

back to top

13.1.14 Gate field (compensating charge)

These variables are relevant to gate field (compensating charge) Detailed introduction

gate_flag

- **Type**: Boolean
- **Description**: Controls the addition of compensating charge by a charged plate for charged cells.
  - true: A charged plate is placed at the **zgate** position to add compensating charge. The direction is determined by **efield_dir**.
  - false: No compensating charge is added.
- **Default**: false

zgate

- **Type**: Real
- **Description**: position of the charged plate in the unit cell
- **Unit**: Unit cell size
- **Default**: 0.5
- **Constraints**: 0 <= **zgate** < 1

block

- **Type**: Boolean
- **Description**: Controls the addition of a potential barrier to prevent electron spillover.
  - true: A potential barrier is added from **block_down** to **block_up** with a height of **block_height**. If **dip_cor_flag** is set to true, **efield_pos_dec** is used to smoothly increase and decrease the potential barrier.
  - false: No potential barrier is added.
- **Default**: false

block_down

- **Type**: Real
- **Description**: lower beginning of the potential barrier
- **Unit**: Unit cell size
- **Default**: 0.45
- **Constraints**: 0 <= **block_down** < **block_up** < 1

block_up

- **Type**: Real
- **Description**: upper beginning of the potential barrier
- **Unit**: Unit cell size
- **Default**: 0.55
- **Constraints**: 0 <= **block_down** < **block_up** < 1

block_height

- **Type**: Real
- **Description**: height of the potential barrier
- **Unit**: Rydberg
- **Default**: 0.1

back to top

13.1.15 Exact Exchange

These variables are relevant when using hybrid functionals.

Availablity: dft_functional==*hse/hf/pbe0/scan0/opt_orb* or rpa==*True*, and basis_type==*lcao/lcao_in_pw*

exx_hybrid_alpha

- **Type**: Real
- **Description**: fraction of Fock exchange in hybrid functionals, so that $E_X = \alpha E_X + (1-\alpha)E_{X,\text{LDA/GGA}}$
- **Default**:
  - 1: if dft_functional==*hf*
  - 0.25: else

exx_hse_omega

- **Type**: Real
- **Description**: range-separation parameter in HSE functional, such that $1/r = \text{erfc}(\omega r)/r + \text{erf}(\omega r)/r$
- **Default**: 0.11

exx_separate_loop

- **Type**: Boolean
- **Description**: There are two types of iterative approaches provided by ABACUS to evaluate Fock exchange.
  - False: Start with a GGA-Loop, and then Hybrid-Loop, in which EXX Hamiltonian H_{exx} is updated with electronic iterations.
  - True: A two-step method is employed, i.e. in the inner iterations, density matrix is updated, while in the outer iterations, H_{exx} is calculated based on density matrix that converges in the inner iteration.
- **Default**: True

exx_hybrid_step

- **Type**: Integer
- **Availability**: exx_separate_loop==*1*
- **Description**: the maximal iteration number of the outer-loop, where the Fock exchange is calculated
- **Default**: 100

exx_mixing_beta

- **Type**: Real
- **Availability**: exx_separate_loop==*1*
- **Description**: mixing_beta for densty matrix in each iteration of the outer-loop
- **Default**: 1.0

exx_lambda

- **Type**: Real
- **Availability**: basis_type==*lcao_in_pw*
- **Description**: It is used to compensate for divergence points at G=0 in the evaluation of Fock exchange using *lcao_in_pw* method.
- **Default**: 0.3

exx_pca_threshold

- **Type**: Real
- **Description**: To accelerate the evaluation of four-center integrals $(ik|jl)$, the product of atomic orbitals are expanded in the basis of auxiliary basis functions (ABF): $\Phi_i\Phi_j \sim C_{ij}^k P_k$. The size of the ABF (i.e. number of P_k) is reduced using principal component analysis. When a large PCA threshold is used, the number of ABF will be reduced, hence the calculation becomes faster. However, this comes at the cost of computational accuracy. A relatively safe choice of the value is 1e-4.
- **Default**: 1E-4

exx_c_threshold

- **Type**: Real
- **Description**: See also the entry *exx_pca_threshold*. Smaller components (less than exx_c_threshold) of the C_{ij}^k matrix are neglected to accelerate calculation. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-4.
- **Default**: 1E-4

exx_v_threshold

- **Type**: Real
- **Description**: See also the entry *exx_pca_threshold*. With the approximation $\Phi_i\Phi_j \sim C_{ij}^k P_k$, the four-center integral in Fock exchange is expressed as $(ik|jl) = \Sigma_{a,b} C_{ij}^a V_{ab} C_{kl}^b$, where $V_{ab} = (P_a|P_b)$ is a double-center integral. Smaller values of the V matrix can be truncated to accelerate calculation. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 0, i.e. no truncation.
- **Default**: 1E-1

exx_dm_threshold

- **Type**: Real
- **Description**: The Fock exchange can be expressed as $\Sigma_{k,l}(ik|jl)D_{kl}$ where D is the density matrix. Smaller values of the density matrix can be truncated to accelerate calculation. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-4.
- **Default**: 1E-4

exx_c_grad_threshold

- **Type**: Real
- **Description**: See also the entry *exx_pca_threshold*. ∇C_{ij}^{k} is used in force and stress. Smaller components (less than exx_c_grad_threshold) of the ∇C_{ij}^{k} matrix are neglected to accelerate calculation. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-4.
- **Default**: 1E-4

exx_v_grad_threshold

- **Type**: Real
- **Description**: See also the entry *exx_pca_threshold*. With the approximation $\Phi_i\Phi_j \sim C_{ij}^{k}P_k$, the four-center integral in Fock exchange is expressed as $(ik|jl) = \Sigma_{a,b}C_{ij}^{a}V_{ab}C_{kl}^{b}$, where $V_{ab} = (P_a|P_b)$ is a double-center integral. ∇V_{ab} is used in force and stress. Smaller values of the V matrix can be truncated to accelerate calculation. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 0, i.e. no truncation.
- **Default**: 1E-1

exx_schwarz_threshold

- **Type**: Real
- **Description**: In practice the four-center integrals are sparse, and using Cauchy-Schwartz inequality, we can find an upper bound of each integral before carrying out explicit evaluations. Those that are smaller than exx_schwarz_threshold will be truncated. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-5. (Currently not used)
- **Default**: 0

exx_cauchy_threshold

- **Type**: Real
- **Description**: In practice the Fock exchange matrix is sparse, and using Cauchy-Schwartz inequality, we can find an upper bound of each matrix element before carrying out explicit evaluations. Those that are smaller than exx_cauchy_threshold will be truncated. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-7.
- **Default**: 1E-7

exx_cauchy_force_threshold

- **Type**: Real
- **Description**: In practice the Fock exchange matrix in force is sparse, and using Cauchy-Schwartz inequality, we can find an upper bound of each matrix element before carrying out explicit evaluations. Those that are smaller than exx_cauchy_force_threshold will be truncated. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-7.
- **Default**: 1E-7

exx_cauchy_stress_threshold

- **Type**: Real
- **Description**: In practice the Fock exchange matrix in stress is sparse, and using Cauchy-Schwartz inequality, we can find an upper bound of each matrix element before carrying out explicit evaluations. Those that are smaller than exx_cauchy_stress_threshold will be truncated. The larger the threshold is, the faster the calculation and the lower the accuracy. A relatively safe choice of the value is 1e-7.
- **Default**: 1E-7

exx_ccp_threshold

- **Type**: Real
- **Description**: It is related to the cutoff of on-site Coulomb potentials. (Currently not used)
- **Default**: 1e-8

exx_ccp_rmesh_times

- **Type**: Real
- **Description**: This parameter determines how many times larger the radial mesh required for calculating Columb potential is to that of atomic orbitals. For HSE, setting it to 1 is enough. But for PBE0, a much larger number must be used.
- **Default**:
  - 1.5: if dft_functional==*hse*
  - 5: else

exx_distribute_type

- **Type**: String
- **Description**: When running in parallel, the evaluation of Fock exchange is done by distributing atom pairs on different processes, then gather the results. exx_distribute_type governs the mechanism of distribution. Available options are `htime`, `order`, `kmean1` and `kmeans2`.
  - `order`: Atom pairs are simply distributed by their orders.
  - `htime`: The balance in time is achieved on each processor, hence if the memory is sufficient, this is the recommended method.
  - `kmeans1` , `kmeans2`: Two methods where the k-means clustering method is used to reduce memory requirement. They might be necessary for very large systems. (Currently not used)
- **Default**: `htime`

exx_opt_orb_lmax

- **Type**: Integer
- **Availability**: dft_functional==*opt_orb*
- **Description**: The maximum l of the spherical Bessel functions, when the radial part of opt-ABFs are generated as linear combinations of spherical Bessel functions. A reasonable choice is 2.
- **Default**: 0

exx_opt_orb_ecut

- **Type**: Real
- **Availability**: dft_functional==*opt_orb*
- **Description**: The cut-off of plane wave expansion, when the plane wave basis is used to optimize the radial ABFs. A reasonable choice is 60.
- **Default**: 0
- **Unit**: Ry

exx_opt_orb_tolerence

- **Type**: Real
- **Availability**: dft_functional==*opt_orb*
- **Description**: The threshold when solving for the zeros of spherical Bessel functions. A reasonable choice is 1e-12.
- **Default**: 0

exx_real_number

- **Type**: Boolean
- **Description**:
  - True: Enforce LibRI to use `double` data type.
  - False: Enforce LibRI to use `complex` data type.
- **Default**: depends on the *gamma_only* option
  - True: if gamma_only
  - False: else

rpa_ccp_rmesh_times

- **Type**: Real
- **Description**: How many times larger the radial mesh required is to that of atomic orbitals in the postprocess calculation of the **bare** Coulomb matrix for RPA, GW, etc.
- **Default**: 10

back to top

13.1.16 Molecular dynamics

These variables are used to control molecular dynamics calculations. For more information, please refer to *md.md* in detail.

md_type

- **Type**: String
- **Description**: Control the algorithm to integrate the equation of motion for molecular dynamics (MD), see *md.md* in detail.
  - fire: a MD-based relaxation algorithm, named fast inertial relaxation engine.
  - nve: NVE ensemble with velocity Verlet algorithm.
  - nvt: NVT ensemble, see *md_thermostat* in detail.
  - npt: Nose-Hoover style NPT ensemble, see *md_pmode* in detail.
  - langevin: NVT ensemble with Langevin thermostat, see *md_damp* in detail.
  - msst: MSST method, see *msst_direction*, *msst_vel*, *msst_qmass*, *msst_vis*, *msst_tscale* in detail.
- **Default**: nvt

md_nstep

- **Type**: Integer
- **Description**: The total number of molecular dynamics steps.
- **Default**: 10

md_dt

- **Type**: Real
- **Description**: The time step used in molecular dynamics calculations.
- **Default**: 1.0
- **Unit**: fs

md_thermostat

- **Type**: String
- **Description**: Specify the temperature control method used in NVT ensemble.
  - nhc: Nose-Hoover chain, see *md_tfreq* and *md_tchain* in detail.
  - anderson: Anderson thermostat, see *md_nraise* in detail.
  - berendsen: Berendsen thermostat, see *md_nraise* in detail.
  - rescaling: velocity Rescaling method 1, see *md_tolerance* in detail.
  - rescale_v: velocity Rescaling method 2, see *md_nraise* in detail.
- **Default**: nhc

md_tfirst, md_tlast

- **Type**: Real
- **Description**: The temperature used in molecular dynamics calculations.

  If `md_tfirst` is unset or less than zero, *init_vel* is autoset to be `true`. If *init_vel* is `true`, the initial temperature will be determined by the velocities read from `STRU`. In this case, if velocities are unspecified in `STRU`, the initial temperature is set to zero.

  If `md_tfirst` is set to a positive value and *init_vel* is `true` simultaneously, please make sure they are consistent, otherwise abacus will exit immediately.

  Note that `md_tlast` is only used in NVT/NPT simulations. If `md_tlast` is unset or less than zero, `md_tlast` is set to `md_tfirst`. If `md_tlast` is set to be different from `md_tfirst`, ABACUS will automatically change the temperature from `md_tfirst` to `md_tlast`.
- **Default**: No default
- **Unit**: K

md_restart

- **Type**: Boolean
- **Description**: Control whether to restart molecular dynamics calculations and time-dependent density functional theory calculations.
  - True: ABACUS will read in `${read_file_dir}/Restart_md.dat` to determine the current step `${md_step}`, then read in the corresponding `STRU_MD_${md_step}` in the folder `OUT.$suffix/STRU/` automatically. For tddft, ABACUS will also read in `LOWF_K_${kpoint}` of the last step (You need to set out_wfc_lcao=1 and out_app_flag=0 to obtain this file).
  - False: ABACUS will start molecular dynamics calculations normally from the first step.
- **Default**: False

md_restartfreq

- **Type**: Integer
- **Description**: The output frequency of `OUT.${suffix}/Restart_md.dat` and structural files in the directory `OUT.${suffix}/STRIU/`, which are used to restart molecular dynamics calculations, see *md_restart* in detail.
- **Default**: 5

md_dumpfreq

- **Type**: Integer
- **Description**: The output frequency of `OUT.${suffix}/MD_dump` in molecular dynamics calculations, which including the information of lattices and atoms.
- **Default**: 1

dump_force

- **Type**: Boolean
- **Description**: Whether to output atomic forces into the file `OUT.${suffix}/MD_dump`.
- **Default**: True

dump_vel

- **Type**: Boolean
- **Description**: Whether to output atomic velocities into the file `OUT.${suffix}/MD_dump`.
- **Default**: True

dump_virial

- **Type**: Boolean
- **Description**: Whether to output lattice virials into the file `OUT.${suffix}/MD_dump`.
- **Default**: True

md_seed

- **Type**: Integer
- **Description**: The random seed to initialize random numbers used in molecular dynamics calculations.
  - < 0: No srand() function is called.
  - >= 0: The function srand(md_seed) is called.
- **Default**: -1

md_tfreq

- **Type**: Real
- **Description**: Control the frequency of temperature oscillations during the simulation. If it is too large, the temperature will fluctuate violently; if it is too small, the temperature will take a very long time to equilibrate with the atomic system.

  Note: It is a system-dependent empirical parameter, ranging from 1/(40*md_dt) to 1/(100*md_dt). An improper choice might lead to the failure of jobs.
- **Default**: 1/40/md_dt
- **Unit**: fs^{-1}

md_tchain

- **Type**: Integer
- **Description**: Number of thermostats coupled with the particles in the NVT/NPT ensemble based on the Nose-Hoover style non-Hamiltonian equations of motion.
- **Default**: 1

md_pmode

- **Type**: String
- **Description**: Specify the cell fluctuation mode in NPT ensemble based on the Nose-Hoover style non-Hamiltonian equations of motion.
  - iso: The three diagonal elements of the lattice are fluctuated isotropically.
  - aniso: The three diagonal elements of the lattice are fluctuated anisotropically.
  - tri: The lattice must be a lower-triangular matrix, and all six freedoms are fluctuated.
- **Default**: iso
- **Relavent**: *md_tfreq*, *md_tchain*, *md_pcouple*, *md_pfreq*, and *md_pchain*.

md_prec_level

- **Type**: Integer
- **Description**: Determine the precision level of variable-cell molecular dynamics calculations.
  - 0: FFT grids do not change, only G vectors and K vectors are changed due to the change of lattice vector. This level is suitable for cases where the variation of the volume and shape is not large, and the efficiency is relatively higher.
  - 2: FFT grids change per step. This level is suitable for cases where the variation of the volume and shape is large, such as the MSST method. However, accuracy comes at the cost of efficiency.
- **Default**: 0

ref_cell_factor

- **Type**: Real
- **Description**: Construct a reference cell bigger than the initial cell. The reference cell has to be large enough so that the lattice vectors of the fluctuating cell do not exceed the reference lattice vectors during MD. Typically, 1.02 ~ 1.10 is sufficient. However, the cell fluctuations depend on the specific system and thermodynamic conditions. So users must test for a proper choice. This parameters should be used in conjunction with *erf_ecut*, *erf_height*, and *erf_sigma*.
- **Default**: 1.0

md_pcouple

- **Type**: String
- **Description**: The coupled lattice vectors will scale proportionally in NPT ensemble based on the Nose-Hoover style non-Hamiltonian equations of motion.
  - none: Three lattice vectors scale independently.
  - xyz: Lattice vectors x, y, and z scale proportionally.
  - xy: Lattice vectors x and y scale proportionally.
  - xz: Lattice vectors x and z scale proportionally.
  - yz: Lattice vectors y and z scale proportionally.
- **Default**: none

md_pfirst, md_plast

- **Type**: Real
- **Description**: The target pressure used in NPT ensemble simulations, the default value of `md_plast` is `md_pfirst`. If `md_plast` is set to be different from `md_pfirst`, ABACUS will automatically change the target pressure from `md_pfirst` to `md_plast`.
- **Default**: -1.0
- **Unit**: kbar

md_pfreq

- **Type**: Real
- **Description**: The frequency of pressure oscillations during the NPT ensemble simulation. If it is too large, the pressure will fluctuate violently; if it is too small, the pressure will take a very long time to equilibrate with the atomic system.

  Note: It is a system-dependent empirical parameter. An improper choice might lead to the failure of jobs.
- **Default**: 1/400/md_dt
- **Unit**: kbar^{-1}

md_pchain

- **Type**: Integer
- **Description**: The number of thermostats coupled with the barostat in the NPT ensemble based on the Nose-Hoover style non-Hamiltonian equations of motion.
- **Default**: 1

lj_rcut

- **Type**: Real
- **Description**: Cut-off radius for Leonard Jones potential.
- **Default**: 8.5 (for He)
- **Unit**: Angstrom

lj_epsilon

- **Type**: Real
- **Description**: The value of epsilon for Leonard Jones potential.
- **Default**: 0.01032 (for He)
- **Unit**: eV

lj_sigma

- **Type**: Real
- **Description**: The value of sigma for Leonard Jones potential.
- **Default**: 3.405 (for He)
- **Unit**: Angstrom

pot_file

- **Type**: String
- **Description**: The filename of DP potential files, see *md.md* in detail.
- **Default**: graph.pb

msst_direction

- **Type**: Integer
- **Description**: The direction of the shock wave in the MSST method.
  - 0: x direction
  - 1: y direction
  - 2: z direction
- **Default**: 2

msst_vel

- **Type**: Real
- **Description**: The velocity of the shock wave in the MSST method.
- **Default**: 0.0
- **Unit**: Angstrom/fs

msst_vis

- **Type**: Real
- **Description**: Artificial viscosity in the MSST method.
- **Default**: 0.0
- **Unit**: g/(mol*Angstrom*fs)

msst_tscale

- **Type**: Real
- **Description**: The reduction percentage of the initial temperature used to compress volume in the MSST method.
- **Default**: 0.01

msst_qmass

- **Type**: Real
- **Description**: Inertia of the extended system variable. You should set a number larger than 0.
- **Default**: No default
- **Unit**: $\mathrm{g}^2/(\mathrm{mol}^2 * \mathrm{Angstrom}^4)$

md_damp

- **Type**: Real
- **Description**: The damping parameter used to add fictitious force in the Langevin method.
- **Default**: 1.0
- **Unit**: fs

md_tolerance

- **Type**: Real
- **Description**: Thr temperature tolerance for velocity rescaling. Velocities are rescaled if the current and target temperature differ more than `md_tolerance`.
- **Default**: 100.0
- **Unit**: K

md_nraise

- **Type**: Integer
- **Description**:
  - Anderson: The "collision frequency" parameter is given as 1/`md_nraise`.
  - Berendsen: The "rise time" parameter is given in units of the time step: tau = `md_nraise`*`md_dt`, so `md_dt`/tau = 1/`md_nraise`.
  - Rescale_v: Every `md_nraise` steps the current temperature is rescaled to the target temperature.
- **Default**: 1

cal_syns

- **Type**: Boolean
- **Description**: Whether the asynchronous overlap matrix is calculated for Hefei-NAMD.
- **Default**: False

dmax

- **Type**: Real
- **Description**: The maximum displacement of all atoms in one step. This parameter is useful when *cal_syns* = True.
- **Default**: 0.01
- **Unit**: bohr

back to top

13.1.17 DFT+*U* correction

These variables are used to control DFT+U correlated parameters

dft_plus_u

- **Type**: Integer
- **Description**: Determines whether to calculate the plus U correction, which is especially important for correlated electrons.
  - 1: Calculate plus U correction with radius-adjustable localized projections (with parameter `onsite_radius`).
  - 2: Calculate plus U correction using first zeta of NAOs as projections (this is old method for testing).
  - 0: Do not calculate plus U correction.
- **Default**: 0

orbital_corr

- **Type**: Integer
- **Description**: Specifies which orbits need plus U correction for each atom type ($l_1, l_2, l_3, \ldots$ for atom type 1, 2, 3, respectively).
  - -1: The plus U correction will not be calculated for this atom.
  - 1: For p-electron orbits, the plus U correction is needed.
  - 2: For d-electron orbits, the plus U correction is needed.
  - 3: For f-electron orbits, the plus U correction is needed.
- **Default**: None

hubbard_u

- **Type**: Real
- **Description**: Specifies the Hubbard Coulomb interaction parameter U (eV) in plus U correction, which should be specified for each atom unless the Yukawa potential is used.

  Note: Since only the simplified scheme by Duradev is implemented, the 'U' here is actually U-effective, which is given by Hubbard U minus Hund J.
- **Default**: 0.0

yukawa_potential

- **Type**: Boolean
- **Description**: Determines whether to use the local screen Coulomb potential method to calculate the values of U and J.
  - True: `hubbard_u` does not need to be specified.
  - False: `hubbard_u` does need to be specified.
- **Default**: False

yukawa_lambda

- **Type**: Real
- **Availability**: DFT+U with `yukawa_potential` = True.
- **Description**: The screen length of Yukawa potential. If left to default, the screen length will be calculated as an average of the entire system. It's better to stick to the default setting unless there is a very good reason.
- **Default**: Calculated on the fly.

uramping

- **Type**: Real
- **Unit**: eV
- **Availability**: DFT+U calculations with `mixing_restart > 0`.
- **Description**: Once `uramping` > 0.15 eV. DFT+U calculations will start SCF with U = 0 eV, namely normal LDA/PBE calculations. Once SCF restarts when `drho<mixing_restart`, U value will increase by `uramping` eV. SCF will repeat above calcuations until U values reach target defined in `hubbard_u`. As for `uramping=1.0 eV`, the recommendations of `mixing_restart` is around `5e-4`.
- **Default**: -1.0.

omc

- **Type**: Integer
- **Description**: The parameter controls the form of occupation matrix control used.
  - 0: No occupation matrix control is performed, and the onsite density matrix will be calculated from wavefunctions in each SCF step.
  - 1: The first SCF step will use an initial density matrix read from a file named `[initial_onsite.dm](http://initial_onsite.dm/)`, but for later steps, the onsite density matrix will be updated.
  - 2: The same onsite density matrix from `initial_onsite.dm` will be used throughout the entire calculation.

  Note : The easiest way to create `initial_onsite.dm` is to run a DFT+U calculation, look for a file named `onsite.dm` in the OUT.prefix directory, and make replacements there. The format of the file is rather straight-forward.
- **Default**: 0

onsite_radius

- **Type**: Real
- **Availability**: `dft_plus_u` is set to 1
- **Description**:
  - The `Onsite-radius` parameter facilitates modulation of the single-zeta portion of numerical atomic orbitals for projections for DFT+U.
  - The modulation algorithm includes a smooth truncation applied directly to the tail of the original orbital, followed by normalization. Consider the function: $$g(r;\sigma) = \begin{cases} 1 - \exp\left(-\frac{(r-r_c)^2}{2\sigma^2}\right), & r < r_c \\ 0, & r \geq r_c \end{cases}$$
  - where σ is a parameter that controls the smoothing interval. A normalized function truncated smoothly at r_c can be represented as:

$$\alpha(r) = \frac{\chi(r)g(r;\sigma)}{\langle \chi(r)g(r;\sigma), \chi(r)g(r;\sigma)\rangle}$$

  - To find an appropriate σ, the optimization process is as follows:
  - Maximizing the overlap integral under a normalization constraint is equivalent to minimizing an error function:

$$\min\langle \chi(r) - \alpha(r), \chi(r) - \alpha(r)\rangle \quad \text{subject to} \quad \langle \alpha(r), \alpha(r)\rangle = 1$$

  - Similar to the process of generating numerical atomic orbitals, this optimization choice often induces additional oscillations in the outcome. To suppress these oscillations, we may include a derivative term in the objective function ($f'(r) \equiv \mathrm{d}f(r)/\mathrm{d}r$):

$$\min\left[\gamma\langle \chi(r) - \alpha(r), \chi(r) - \alpha(r)\rangle + \langle \chi'(r) - \alpha'(r), \chi'(r) - \alpha'(r)\rangle\right] \quad \text{subject to} \quad \langle \alpha(r), \alpha(r)\rangle = 1$$

  - where γ is a parameter that adjusts the relative weight of the error function to the derivative error function.
- **Unit**: Bohr
- **Default**: 5.0

back to top

13.1.18 vdW correction

These variables are used to control vdW-corrected related parameters.

vdw_method

- **Type**: String
- **Description**: Specifies the method used for Van der Waals (VdW) correction. Available options are:
  - `d2`: Grimme's D2 dispersion correction method
  - `d3_0`: Grimme's DFT-D3(0) dispersion correction method
  - `d3_bj`: Grimme's DFTD3(BJ) dispersion correction method
  - `none`: no vdW correction
- **Default**: none

vdw_s6

- **Type**: Real
- **Availability**: `vdw_method` is set to `d2`, `d3_0`, or `d3_bj`
- **Description**: This scale factor is used to optimize the interaction energy deviations in van der Waals (vdW) corrected calculations. The recommended values of this parameter are dependent on the chosen vdW correction method and the DFT functional being used. For DFT-D2, the recommended values are 0.75 (PBE), 1.2 (BLYP), 1.05 (B-P86), 1.0 (TPSS), and 1.05 (B3LYP). For DFT-D3, recommended values with different DFT functionals can be found on the here. The default value of this parameter in ABACUS is set to be the recommended value for PBE.
- **Default**:
  - 0.75: if `vdw_method` is set to `d2`
  - 1.0: if `vdw_method` is set to `d3_0` or `d3_bj`

vdw_s8

- **Type**: Real
- **Availability**: `vdw_method` is set to `d3_0` or `d3_bj`
- **Description**: This scale factor is relevant for D3(0) and D3(BJ) van der Waals (vdW) correction methods. The recommended values of this parameter with different DFT functionals can be found on the webpage. The default value of this parameter in ABACUS is set to be the recommended value for PBE.
- **Default**:
  - 0.722: if `vdw_method` is set to `d3_0`
  - 0.7875: if `vdw_method` is set to `d3_bj`

vdw_a1

- **Type**: Real
- **Availability**: `vdw_method` is set to `d3_0` or `d3_bj`
- **Description**: This damping function parameter is relevant for D3(0) and D3(BJ) van der Waals (vdW) correction methods. The recommended values of this parameter with different DFT functionals can be found on the webpage. The default value of this parameter in ABACUS is set to be the recommended value for PBE.
- **Default**:
  - 1.217: if `vdw_method` is set to `d3_0`
  - 0.4289: if `vdw_method` is set to `d3_bj`

vdw_a2

- **Type**: Real
- **Availability**: `vdw_method` is set to `d3_0` or `d3_bj`
- **Description**: This damping function parameter is only relevant for D3(0) and D3(BJ) van der Waals (vdW) correction methods. The recommended values of this parameter with different DFT functionals can be found on the webpage. The default value of this parameter in ABACUS is set to be the recommended value for PBE.
- **Default**:
  - 1.0: if `vdw_method` is set to `d3_0`
  - 4.4407: if `vdw_method` is set to `d3_bj`

vdw_d

- **Type**: Real
- **Availability**: `vdw_method` is set to `d2`
- **Description**: Controls the damping rate of the damping function in the DFT-D2 method.
- **Default**: 20

vdw_abc

- **Type**: Integer
- **Availability**: `vdw_method` is set to `d3_0` or `d3_bj`
- **Description**: Determines whether three-body terms are calculated for DFT-D3 methods.
  - True: ABACUS will calculate the three-body term.
  - False: The three-body term is not included.
- **Default**: False

vdw_C6_file

- **Type**: String
- **Availability**: `vdw_method` is set to `d2`
- **Description**: Specifies the name of the file containing C_6 parameters for each element when using the D2 method. If not set, ABACUS uses the default C_6 parameters (Jnm6/mol) stored in the program. To manually set the C_6 parameters, provide a file containing the parameters. An example is given by:

```
H  0.1
Si 9.0
```

  Namely, each line contains the element name and the corresponding C_6 parameter.
- **Default**: default

vdw_C6_unit

- **Type**: String
- **Availability**: `vdw_C6_file` is not default
- **Description**: Specifies the unit of the provided C_6 parameters in the D2 method. Available options are:
  - `Jnm6/mol` (J·nm^6/mol)
  - `eVA` (eV·Angstrom)
- **Default**: Jnm6/mol

vdw_R0_file

- **Type**: String
- **Availability**: `vdw_method` is set to `d2`
- **Description**: Specifies the name of the file containing R_0 parameters for each element when using the D2 method. If not set, ABACUS uses the default R_0 parameters (Angstrom) stored in the program. To manually set the R_0 parameters, provide a file containing the parameters. An example is given by:

```
Li 1.0
Cl 2.0
```

  Namely, each line contains the element name and the corresponding R_0 parameter.
- **Default**: default

vdw_R0_unit

- **Type**: String
- **Availability**: `vdw_R0_file` is not default
- **Description**: Specifies the unit for the R_0 parameters in the D2 method when manually set by the user. Available options are:
  - `A` (Angstrom)
  - `Bohr`
- **Default**: A

vdw_cutoff_type

- **Type**: String
- **Description**: Determines the method used for specifying the cutoff radius in periodic systems when applying Van der Waals correction. Available options are:
  - `radius`: The supercell is selected within a sphere centered at the origin with a radius defined by `vdw_cutoff_radius`.
  - `period`: The extent of the supercell is explicitly specified using the `vdw_cutoff_period` keyword.
- **Default**: radius

vdw_cutoff_radius

- **Type**: Real
- **Availability**: `vdw_cutoff_type` is set to `radius`
- **Description**: Defines the radius of the cutoff sphere when `vdw_cutoff_type` is set to `radius`. The default values depend on the chosen `vdw_method`.
- **Default**:
  - 56.6918 if `vdw_method` is set to `d2`
  - 95 if `vdw_method` is set to `d3_0` or `d3_bj`
- **Unit**: defined by `vdw_radius_unit` (default `Bohr`)

vdw_radius_unit

- **Type**: String
- **Availability**: `vdw_cutoff_type` is set to `radius`
- **Description**: specify the unit of `vdw_cutoff_radius`. Available options are:
  - `A`(Angstrom)
  - `Bohr`
- **Default**: Bohr

vdw_cutoff_period

- **Type**: Integer Integer Integer
- **Availability**: `vdw_cutoff_type` is set to `period`
- **Description**: The three integers supplied here explicitly specify the extent of the supercell in the directions of the three basis lattice vectors.
- **Default**: 3 3 3

vdw_cn_thr

- **Type**: Real
- **Availability**: `vdw_method` is set to `d3_0` or `d3_bj`
- **Description**: The cutoff radius when calculating coordination numbers.
- **Default**: 40
- **Unit**: defined by `vdw_cn_thr_unit` (default: `Bohr`)

vdw_cn_thr_unit

- **Type**: String
- **Description**: Unit of the coordination number cutoff (`vdw_cn_thr`). Available options are:
  - `A`(Angstrom)
  - `Bohr`
- **Default**: Bohr

back to top

13.1.19 Berry phase and wannier90 interface

These variables are used to control berry phase and wannier90 interface parameters. Detail introduce

berry_phase

- **Type**: Boolean
- **Description**: controls the calculation of Berry phase
  - true: Calculate Berry phase.
  - false: Do not calculate Berry phase.
- **Default**: false

gdir

- **Type**: Integer
- **Description**: the direction of the polarization in the lattice vector for Berry phase calculation
  - 1: Calculate the polarization in the direction of the lattice vector a_1 defined in the STRU file.
  - 2: Calculate the polarization in the direction of the lattice vector a_2 defined in the STRU file.
  - 3: Calculate the polarization in the direction of the lattice vector a_3 defined in the STRU file.
- **Default**: 3

towannier90

- **Type**: Integer
- **Description**: Controls the generation of files for the Wannier90 code.
  - 1: Generate files for the Wannier90 code.
  - 0: Do not generate files for the Wannier90 code.
- **Default**: 0

nnkpfile

- **Type**: String
- **Description**: the file name generated when running "wannier90 -pp …" command
- **Default**: seedname.nnkp

wannier_method

- **Type**: Integer
- **Description**: Only available on LCAO basis, using different methods to generate "*.mmn" file and "*.amn" file.
  - 1: Calculated using the `lcao_in_pw` method, the calculation accuracy can be improved by increasing `ecutwfc` to maintain consistency with the pw basis set results.
  - 2: The overlap between atomic orbitals is calculated using grid integration. The radial grid points are generated using the Gauss-Legendre method, while the spherical grid points are generated using the Lebedev-Laikov method.
- **Default**: 1

wannier_spin

- **Type**: String
- **Description**: the spin direction for the Wannier function calculation when nspin is set to 2
  - `up`: Calculate spin up for the Wannier function.
  - `down`: Calculate spin down for the Wannier function.
- **Default**: `up`

out_wannier_mmn

- **Type**: Bool
- **Description**: write the "*.mmn" file or not.
  - 0: don't write the "*.mmn" file.
  - 1: write the "*.mmn" file.
- **Default**: 1

out_wannier_amn

- **Type**: Bool
- **Description**: write the "*.amn" file or not.
  - 0: don't write the "*.amn" file.
  - 1: write the "*.amn" file.
- **Default**: 1

out_wannier_eig

- **Type**: Bool
- **Description**: write the "*.eig" file or not.
  - 0: don't write the "*.eig" file.
  - 1: write the "*.eig" file.
- **Default**: 1

out_wannier_unk

- **Type**: Bool
- **Description**: write the "UNK.*" file or not.
  - 0: don't write the "UNK.*" file.
  - 1: write the "UNK.*" file.
- **Default**: 0

out_wannier_wvfn_formatted

- **Type**: Bool
- **Description**: write the "UNK.*" file in ASCII format or binary format.
  - 0: write the "UNK.*" file in binary format.
  - 1: write the "UNK.*" file in ASCII format (text file format).
- **Default**: 1

back to top

13.1.20 TDDFT: time dependent density functional theory

td_edm

- **Type**: Integer
- **Description**: the method to calculate the energy density matrix
  - 0: new method (use the original formula).
  - 1: old method (use the formula for ground state).
- **Default**: 0

td_print_eij

- **Type**: Real
- **Description**:
  - <0: don't print E_{ij}.
  - >=0: print the E_{ij} ($< \psi_i|H|\psi_j >$) elements which are larger than td_print_eij.
- **Default**: -1

td_propagator

- **Type**: Integer
- **Description**: method of propagator
  - 0: Crank-Nicolson.
  - 1: 4th Taylor expansions of exponential.
  - 2: enforced time-reversal symmetry (ETRS).
- **Default**: 0

td_vext

- **Type**: Boolean
- **Description**:
  - True: add a laser material interaction (extern laser field).
  - False: no extern laser field.
- **Default**: False

td_vext_dire

- **Type**: String
- **Description**: If `td_vext` is True, the td_vext_dire is a string to set the number of electric fields, like `td_vext_dire 1 2` representing external electric field is added to the x and y axis at the same time. Parameters of electric field can also be written as a string, like `td_gauss_phase 0 1.5707963267948966` representing the Gauss field in the x and y directions has a phase delay of Pi/2. See below for more parameters of electric field.
  - 1: the direction of external light field is along x axis.
  - 2: the direction of external light field is along y axis.
  - 3: the direction of external light field is along z axis.
- **Default**: 1

td_stype

- **Type**: Integer
- **Description**: type of electric field in space domain
  - 0: length gauge.
  - 1: velocity gauge.
- **Default**: 0

td_ttype

- **Type**: Integer
- **Description**: type of electric field in time domain
  - 0: Gaussian type function.
  - 1: Trapezoid function.
  - 2: Trigonometric function.
  - 3: Heaviside function.
  - 4: HHG function.
- **Default**: 0

td_tstart

- **Type**: Integer
- **Description**: number of steps where electric field starts
- **Default**: 1

td_tend

- **Type**: Integer
- **Description**: number of steps where electric field ends
- **Default**: 100

td_lcut1

- **Type**: Real
- **Description**: cut1 of interval in length gauge
  E = E0 , cut1<x<cut2
  E = -E0/(cut1+1-cut2) , x<cut1 or cut2<x<1
- **Default**: 0.05

td_lcut2

- **Type**: Real
- **Description**: cut2 of interval in length gauge
  E = E0 , cut1<x<cut2
  E = -E0/(cut1+1-cut2) , x<cut1 or cut2<x<1
- **Default**: 0.05

td_gauss_freq

- **Type**: Real
- **Description**: frequency (freq) of Gauss type electric field (fs^-1)
  amp*cos(2pi*freq(t-t0)+phase)exp(-(t-t0)^2/2sigma^2)
- **Default**: 22.13

td_gauss_phase

- **Type**: Real
- **Description**: phase of Gauss type electric field
  amp*(2pi*freq(t-t0)+phase)exp(-(t-t0)^2/2sigma^2)
- **Default**: 0.0

td_gauss_sigma

- **Type**: Real
- **Description**: sigma of Gauss type electric field (fs)
  amp*cos(2pi*freq(t-t0)+phase)exp(-(t-t0)^2/2sigma^2)
- **Default**: 30.0

td_gauss_t0

- **Type**: Real
- **Description**: step number of time center (t0) of Gauss type electric field
  amp*cos(2pi*freq(t-t0)+phase)exp(-(t-t0)^2/2sigma^2)
- **Default**: 100

td_gauss_amp

- **Type**: Real
- **Description**: amplitude (amp) of Gauss type electric field (V/Angstrom)
  amp*cos(2pi*freq(t-t0)+phase)exp(-(t-t0)^2/2sigma^2)
- **Default**: 0.25

td_trape_freq

- **Type**: Real
- **Description**: frequency (freq) of Trapezoid type electric field (fs^-1)
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 1.60

td_trape_phase

- **Type**: Real
- **Description**: phase of Trapezoid type electric field
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 0.0

td_trape_t1

- **Type**: Real
- **Description**: step number of time interval 1 (t1) of Trapezoid type electric field
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 1875

td_trape_t2

- **Type**: Real
- **Description**: step number of time interval 2 (t2) of Trapezoid type electric field
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 5625

td_trape_t3

- **Type**: Real
- **Description**: step number of time interval 3 (t3) of Trapezoid type electric field
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 7500

td_trape_amp

- **Type**: Real
- **Description**: amplitude (amp) of Trapezoid type electric field (V/Angstrom)
  E = amp*cos(2pi*freq*t+phase) t/t1 , t<t1
  E = amp*cos(2pi*freq*t+phase) , t1<t<t2
  E = amp*cos(2pi*freq*t+phase) (1-(t-t2)/(t3-t2)) , t2<t<t3
  E = 0 , t>t3
- **Default**: 2.74

td_trigo_freq1

- **Type**: Real
- **Description**: frequency 1 (freq1) of Trigonometric type electric field (fs^-1)
  amp*cos(2*pi*freq1*t+phase1)*sin(2*pi*freq2*t+phase2)^2
- **Default**: 1.164656

td_trigo_freq2

- **Type**: Real
- **Description**: frequency 2 (freq2) of Trigonometric type electric field (fs^-1)
  amp*cos(2*pi*freq1*t+phase1)*sin(2*pi*freq2*t+phase2)^2
- **Default**: 0.029116

td_trigo_phase1

- **Type**:Real
- **Description**: phase 1 (phase1) of Trigonometric type electric field
  amp*cos(2*pi*freq1*t+phase1)*sin(2*pi*freq2*t+phase2)^2
- **Default**: 0.0

td_trigo_phase2

- **Type**: Real
- **Description**: phase 2 (phase2) of Trigonometric type electric field
  amp*cos(2*pi*freq1*t+phase1)*sin(2*pi*freq2*t+phase2)^2
- **Default**: 0.0

td_trigo_amp

- **Type**: Real
- **Description**: amplitude (amp) of Trigonometric type electric field (V/Angstrom)
  amp*cos(2*pi*freq1*t+phase1)*sin(2*pi*freq2*t+phase2)^2
- **Default**: 2.74

td_heavi_t0

- **Type**: Real
- **Description**: step number of switch time (t0) of Heaviside type electric field
  E = amp , t<t0
  E = 0.0 , t>t0
- **Default**: 100

td_heavi_amp

- **Type**: Real
- **Description**: amplitude (amp) of Heaviside type electric field (V/Angstrom)
  E = amp , t<t0
  E = 0.0 , t>t0
- **Default**: 2.74

out_dipole

- **Type**: Boolean
- **Description**:
  - True: output dipole.
  - False: do not output dipole.
- **Default**: False

out_efield

- **Type**: Boolean
- **Description**: output TDDFT Efield or not(V/Angstrom)
  - True: output efield.
  - False: do not output efield.
- **Default**: False

ocp

- **Type**: Boolean
- **Availability**:
  - For PW and LCAO codes. if set to 1, occupations of bands will be setting of "ocp_set".
  - For TDDFT in LCAO codes. if set to 1, occupations will be constrained since second ionic step.
  - For OFDFT, this feature can't be used.
- **Description**:
- True: fix the occupations of bands.
- False: do not fix the occupations of bands.
- **Default**: False

ocp_set

- **Type**: String
- **Description**: If ocp is True, the ocp_set is a string to set the number of occupancy, like '1 10 * 1 0 1' representing the 13 band occupancy, 12th band occupancy 0 and the rest 1, the code is parsing this string into an array through a regular expression.
- **Default**: none

back to top

13.1.21 Variables useful for debugging

t_in_h

- **Type**: Boolean
- **Description**: Specify whether to include kinetic term in obtaining the Hamiltonian matrix.
  - 0: No.
  - 1: Yes.
- **Default**: 1

vl_in_h

- **Type**: Boolean
- **Description**: Specify whether to include local pseudopotential term in obtaining the Hamiltonian matrix.
  - 0: No.
  - 1: Yes.
- **Default**: 1

vnl_in_h

- **Type**: Boolean
- **Description**: Specify whether to include non-local pseudopotential term in obtaining the Hamiltonian matrix.
  - 0: No.
  - 1: Yes.
- **Default**: 1

vh_in_h

- **Type**: Boolean
- **Description**: Specify whether to include Hartree potential term in obtaining the Hamiltonian matrix.
  - 0: No.
  - 1: Yes.
- **Default**: 1

vion_in_h

- **Type**: Boolean
- **Description**: Specify whether to include local ionic potential term in obtaining the Hamiltonian matrix.
  - 0: No.
  - 1: Yes.
- **Default**: 1

test_force

- **Type**: Boolean
- **Description**: Specify whether to output the detailed components in forces.
  - 0: No.
  - 1: Yes.
- **Default**: 0

test_stress

- **Type**: Boolean
- **Description**: Specify whether to output the detailed components in stress.
  - 0: No.
  - 1: Yes.
- **Default**: 0

colour

- **Type**: Boolean
- **Description**: Specify whether to set the colorful output in terminal.
  - 0: No.
  - 1: Yes.
- **Default**: 0

test_skip_ewald

- **Type**: Boolean
- **Description**: Specify whether to skip the calculation of the ewald energy.
  - 0: No.
  - 1: Yes.
- **Default**: 0

back to top

13.1.22 Electronic conductivities

Frequency-dependent electronic conductivities can be calculated with Kubo-Greenwood formula [Phys. Rev. B 83, 235120 (2011)].

Onsager coefficients:

$L_{mn}(\omega) = (-1)^{m+n} \frac{2\pi e^2\hbar^2}{3m_e^2\omega\Omega}$

$\times \sum_{ij\alpha\mathbf{k}} W(\mathbf{k}) \left(\frac{\epsilon_{i\mathbf{k}}+\epsilon_{j\mathbf{k}}}{2} - \mu\right)^{m+n-2} \times |\langle\Psi_{i\mathbf{k}}|\nabla_\alpha|\Psi_{j\mathbf{k}}\rangle|^2$

$\times[f(\epsilon_{i\mathbf{k}}) - f(\epsilon_{j\mathbf{k}})]\delta(\epsilon_{j\mathbf{k}} - \epsilon_{i\mathbf{k}} - \hbar\omega).$

They can also be computed by j-j correlation function.

$L_{mn} = \frac{2e^{m+n-2}}{3\Omega\hbar\omega}\Im[\tilde{C}_{mn}(\omega)]$ Guassian smearing: $\tilde{C}_{mn} = \int_0^\infty C_{mn}(t)e^{-i\omega t}e^{-\frac{1}{2}s^2t^2}dt$ Lorentzian smearing: $\tilde{C}_{mn} = \int_0^\infty C_{mn}(t)e^{-i\omega t}e^{-\gamma t}dt$

$C_{mn}(t) = -2\theta(t)\Im\left\{Tr\left[\sqrt{\hat{f}}\hat{j}_m(1-\hat{f})e^{i\frac{\hat{H}}{\hbar}t}\hat{j}_n e^{-i\frac{\hat{H}}{\hbar}t}\sqrt{\hat{f}}\right]\right\},$

where j_1 is electric flux and j_2 thermal flux.

Frequency-dependent electric conductivities: $\sigma(\omega) = L_{11}(\omega)$.

Frequency-dependent thermal conductivities: $\kappa(\omega) = \frac{1}{e^2 T}\left(L_{22} - \frac{L_{12}^2}{L_{11}}\right)$.

DC electric conductivities: $\sigma = \lim_{\omega \to 0} \sigma(\omega)$.

Thermal conductivities: $\kappa = \lim_{\omega \to 0} \kappa(\omega)$.

cal_cond

- **Type**: Boolean
- **Availability**: *basis_type* = `pw`
- **Description**: Whether to calculate electronic conductivities.
- **Default**: False

cond_che_thr

- **Type**: Real
- **Availability**: *esolver_type* = `sdft`
- **Description**: Control the error of Chebyshev expansions for conductivities.
- **Default**: 1e-8

cond_dw

- **Type**: Real
- **Availability**: *basis_type* = `pw`
- **Description**: Frequency interval (dω) for frequency-dependent conductivities.
- **Default**: 0.1
- **Unit**: eV

cond_wcut

- **Type**: Real
- **Availability**: *basis_type* = `pw`
- **Description**: Cutoff frequency for frequency-dependent conductivities.
- **Default**: 10.0
- **Unit**: eV

cond_dt

- **Type**: Real
- **Availability**: *basis_type* = `pw`
- **Description**: Time interval (dt) to integrate Onsager coefficients.
- **Default**: 0.02
- **Unit**: a.u.

cond_dtbatch

- **Type**: Integer
- **Availability**: *esolver_type* = `sdft`
- **Description**: exp(iH*dt*cond_dtbatch) is expanded with Chebyshev expansion to calculate conductivities. It is faster but costs more memory.
  - If `cond_dtbatch = 0`: Autoset this parameter to make expansion orders larger than 100.
- **Default**: 0

cond_smear

- **Type**: Integer
- **Description**: Smearing method for conductivities
  - 1: Gaussian smearing
  - 2: Lorentzian smearing
- **Default**: 1

cond_fwhm

- **Type**: Real
- **Availability**: *basis_type* = `pw`
- **Description**: FWHM for conductivities. For Gaussian smearing, FWHM $= 2\sqrt{2\ln 2}s$; for Lorentzian smearing, FWHM $= 2\gamma$.
- **Default**: 0.4
- **Unit**: eV

cond_nonlocal

- **Type**: Boolean
- **Availability**: *basis_type* = `pw`
- **Description**: Whether to consider nonlocal potential correction when calculating velocity matrix $\psi_i \hat{v} \psi_j$.
  - True: $m\hat{v} = \hat{p} + \frac{im}{\hbar}[\hat{V}_{NL}, \hat{r}]$.
  - False: $m\hat{v} \approx \hat{p}$.
- **Default**: True

back to top

13.1.23 Implicit solvation model

These variables are used to control the usage of implicit solvation model. This approach treats the solvent as a continuous medium instead of individual "explicit" solvent molecules, which means that the solute is embedded in an implicit solvent and the average over the solvent degrees of freedom becomes implicit in the properties of the solvent bath.

imp_sol

- **Type**: Boolean
- **Description**: calculate implicit solvation correction
- **Default**: False

eb_k

- **Type**: Real
- **Availability**: `imp_sol` is true.
- **Description**: the relative permittivity of the bulk solvent, 80 for water
- **Default**: 80

tau

- **Type**: Real
- **Description**: The effective surface tension parameter that describes the cavitation, the dispersion, and the repulsion interaction between the solute and the solvent which are not captured by the electrostatic terms
- **Default**: 1.0798e-05
- **Unit**: $Ry/Bohr^2$

sigma_k

- **Type**: Real
- **Description**: the width of the diffuse cavity that is implicitly determined by the electronic structure of the solute
- **Default**: 0.6

nc_k

- **Type**: Real
- **Description**: the value of the electron density at which the dielectric cavity forms
- **Default**: 0.00037
- **Unit**: $Bohr^{-3}$

back to top

13.1.24 Deltaspin

These variables are used to control the usage of deltaspin functionality.

sc_mag_switch

- **Type**: boolean
- **Description**: the switch of deltaspin functionality
  - 0: no deltaspin
  - 1: use the deltaspin method to constrain atomic magnetic moments
- **Default**: 0

decay_grad_switch

- **Type**: boolean
- **Description**: the switch of decay gradient method
  - 0: no decay gradient method
  - 1: use the decay gradient method and set ScDecayGrad in the file specified by `sc_file`. ScDecayGrad is an element dependent parameter, which is used to control the decay rate of the gradient of the magnetic moment.
- **Default**: 0

sc_thr

- **Type**: Real
- **Description**: the threshold of the spin constraint atomic magnetic moment
- **Default**: 1e-6
- **Unit**: Bohr Mag (\muB)

nsc

- **Type**: Integer
- **Description**: the maximum number of steps in the inner lambda loop
- **Default**: 100

nsc_min

- **Type**: Integer
- **Description**: the minimum number of steps in the inner lambda loop
- **Default**: 2

sc_scf_nmin

- **Type**: Integer
- **Description**: the minimum number of outer scf loop before initializing lambda loop
- **Default**: 2

alpha_trial

- **Type**: Real
- **Description**: initial trial step size for lambda in eV/uB^2
- **Default**: 0.01
- **Unit**: eV/uB^2

sccut

- **Type**: Real
- **Description**: restriction of step size in eV/uB
- **Default**: 3
- **Unit**: eV/uB

sc_file

- **Type**: String
- **Description**: the file in json format to specify atomic constraining parameters. An example of the sc_file json file is shown below for the `nspin 4` case:

```
[
    {
        "element": "Fe",
        "itype": 0,
        "ScDecayGrad": 0.9,
        "ScAtomData": [
            {
                "index": 0,
                "lambda": [0, 0, 0],
                "target_mag": [2.0, 0.0, 0.0],
                "constrain": [1,1,1]
            },
            {
                "index": 1,
                "lambda": [0, 0, 0],
                "target_mag_val": 2.0,
                "target_mag_angle1": 80.0,
                "target_mag_angle2": 0.0,
                "constrain": [1,1,1]
            }
        ]
    }
]
```

and

```
[
    {
        "element": "Fe",
        "itype": 0,
        "ScDecayGrad": 0.9,
        "ScAtomData": [
            {
                "index": 0,
                "lambda": 0.0,
                "target_mag": 2.0,
                "constrain": 1
            },
            {
                "index": 1,
                "lambda": 0,
                "target_mag": 2.0,
                "constrain": 1
            }
        ]
    }
]
```

for `nspin 2` case. The difference is that `lambda`, `target_mag`, and `constrain` are scalars in `nspin 2` case, and are vectors in `nspin 4` case.

- **Default**: none

back to top

13.1.25 Quasiatomic Orbital (QO) analysis

These variables are used to control the usage of QO analysis. QO further compress information from LCAO: usually PW basis has dimension in million, LCAO basis has dimension below thousand, and QO basis has dimension below hundred.

qo_switch

- **Type**: Boolean
- **Description**: whether to let ABACUS output QO analysis required files
- **Default**: 0

qo_basis

- **Type**: String
- **Description**: specify the type of atomic basis
  - `pswfc`: use the pseudowavefunction in pseudopotential files as atomic basis. To use this option, please make sure in pseudopotential file there is pswfc in it.
  - `hydrogen`: generate hydrogen-like atomic basis (or with Slater screening).
  - `szv`: use the first set of zeta for each angular momentum from numerical atomic orbitals as atomic basis.

  warning: to use `pswfc` *, please use norm-conserving pseudopotentials with pseudowavefunctions, SG15 pseudopotentials cannot support this option. Developer notes: for ABACUS-lcao calculation, it is the most recommend to use* `szv` *instead of* `pswfc` *which is originally put forward in work of QO implementation on PW basis. The information loss always happens if* `pswfc` *or* `hydrogen` *orbitals are not well tuned, although making kpoints sampling more dense will mitigate this problem, but orbital-adjust parameters are needed to test system-by-system in this case.*
- **Default**: `szv`

qo_strategy

- **Type**: String [String…](optional)
- **Description**: specify the strategy to generate radial orbitals for each atom type. If one parameter is given, will apply to all atom types. If more than one parameters are given but fewer than number of atom type, those unspecified atom type will use default value.

  For `qo_basis hydrogen`
  - `minimal-nodeless`: according to principle quantum number of the highest occupied state, generate only nodeless orbitals, for example Cu, only generate 1s, 2p, 3d and 4f orbitals (for Cu, 4s is occupied, thus $n_{max} = 4$)
  - `minimal-valence`: according to principle quantum number of the highest occupied state, generate only orbitals with highest principle quantum number, for example Cu, only generate 4s, 4p, 4d and 4f orbitals.
  - `full`: similarly according to the maximal principle quantum number, generate all possible orbitals, therefore for Cu, for example, will generate 1s, 2s, 2p, 3s, 3p, 3d, 4s, 4p, 4d, 4f.
  - `energy-full`: will generate hydrogen-like orbitals according to Aufbau principle. For example the Cu (1s2 2s2 2p6 3s2 3p6 3d10 4s1), will generate these orbitals.

  - `energy-valence`: from the highest n (principal quantum number) layer and n-1 layer, generate all occupied and possible ls (angular momentum quantum number) for only once, for example Cu, will generate 4s, 3d and 3p orbitals.

  For `qo_basis pswfc` and `qo_basis szv`

  - `all`: use all possible pseudowavefunctions/numerical atomic orbital (of first zeta) in pseudopotential/numerical atomic orbital file.
  - `s/p/d/...`: only use s/p/d/f/...-orbital(s).
  - `spd`: use s, p and d orbital(s). Any unordered combination is acceptable.

  warning: for `qo_basis hydrogen` *to use* `full`, *generation strategy may cause the space spanned larger than the one spanned by numerical atomic orbitals, in this case, must filter out orbitals in some way*

- **Default**: for `hydrogen`: `energy-valence`, for `pswfc` and `szv`: `all`

qo_screening_coeff

- **Type**: Real [Real...](optional)
- **Description**: rescale the shape of radial orbitals, available for both `qo_basis hydrogen` and `qo_basis pswfc`. cases but has different meaning.

  For `qo_basis pswfc`
  For each atom type, screening factor $e^{-\eta|\mathbf{r}|}$ is multiplied to the pswfc to mimic the behavior of some kind of electron. η is the screening coefficient. If only one value is given, then will apply to each atom type. If not enough values are given, will apply default value to rest of atom types. This parameter plays important role in controlling the spread of QO orbitals together with `qo_thr`.

  For `qo_basis hydrogen`
  If any float number is given, will apply Slater screening to all atom types. Slater screening is a classic and empirical method roughly taking many-electron effect into account for obtaining more accurate results when evaluating electron affinity and ionization energy. The Coulomb potential then becomes $V(r) = -\frac{Z-\sigma}{r}$. For example the effective nuclear charge for Cu 3d electrons now reduces from 29 to 7.85, 4s from 29 to 3.70, which means Slater screening will bring about longer tailing effect. If no value is given, will not apply Slater screening.

- **Default**: 0.1
- **Unit**: Bohr^-1

qo_thr

- **Type**: Real
- **Description**: the convergence threshold determining the cutoff of generated orbital. Lower threshold will yield orbital with larger cutoff radius.
- **Default**: 1.0e-6

back to top

13.2 The STRU file

13.2.1 Examples

The `STRU` file contains the information about the lattice geometry, the name(s) and/or location(s) of the pseudopotential and numerical orbital files, as well as the structural information about the system. We supply two ways of specifying the lattice geometry. Below are two examples of the `STRU` file for the same system:

No latname

For this example, no need to supply any input to the variable `latname` in the INPUT file. (See *input parameters*.)

```
ATOMIC_SPECIES
Si 28.00 Si_ONCV_PBE-1.0.upf upf201 // label; mass; pseudo_file; pseudo_type

NUMERICAL_ORBITAL
Si_gga_8au_60Ry_2s2p1d.orb //numerical_orbital_file

LATTICE_CONSTANT
10.2 // lattice scaling factor (Bohr)

LATTICE_VECTORS
0.5 0.5 0.0 // latvec1
0.5 0.0 0.5 // latvec2
0.0 0.5 0.5 // latvec3

ATOMIC_POSITIONS
Direct //Cartesian or Direct coordinate.
Si // Element type
0.0 // magnetism(Be careful: value 1.0 refers to 1.0 bohr mag, but not fully spin up !
↪!!)
2 // number of atoms
0.00 0.00 0.00 0 0 0
0.25 0.25 0.25 1 1 1
```

latname fcc

We see that this example is a silicon fcc lattice. Apart from setting the lattice vectors manually, we also provide another solution where only the Bravais lattice type is required, and the lattice vectors will be generated automatically. For this example, we need to set `latname="fcc"` in the INPUT file. (See *input parameters*.) And the `STRU` file becomes:

```
ATOMIC_SPECIES
Si 28.00 Si_ONCV_PBE-1.0.upf // label; mass; pseudo_file

NUMERICAL_ORBITAL
Si_gga_8au_60Ry_2s2p1d.orb //numerical_orbital_file

LATTICE_CONSTANT
10.2 // lattice scaling factor (Bohr)

ATOMIC_POSITIONS
Direct //Cartesian or Direct coordinate.
Si // Element type
0.0 // magnetism
2 // number of atoms
0.00 0.00 0.00 0 0 0//the position of atoms and other parameter specify by key word
0.25 0.25 0.25 1 1 1
```

The LATTICE_VECTORS section is removed.

13.2.2 Structure of the file

The `STRU` file contains several sections, and each section must start with a keyword like `ATOMIC_SPECIES`, `NUMERICAL_ORBITAL`, or `LATTICE_CONSTANT`, etc. to signify what type of information that comes below.

ATOMIC_SPECIES

This section provides information about the type of chemical elements contained the unit cell. Each line defines one type of element. The user should specify the name, the mass, and the pseudopotential file used for each element. The mass of the elment is only used in molecular dynamics simulations. For electronic-structure calculations, the actual mass value isn't important. In the above example, we see information is provided for the element `Si`:

```
Si 28.00 Si_ONCV_PBE-1.0.upf upf201 // label; mass; pseudo_file; pseudo_type
```

Here `Si_ONCV_PBE-1.0.upf` is the pseudopotential file. When the path is not specified, the file is assumed to be located in work directory. Otherwise, please explicitly specify the location of the pseudopotential files.

After the pseudopotential file, `upf201` is the type of pseudopotential. There are five options: `upf`(.UPF format), `upf201`(the new .UPF format), `vwr`(.vwr format), `blps`(bulk-derived local pseudopotential), and `auto`(automatically identified). If no pseudopotential type is assigned, the default value is `auto`, and the pseudopotential type will be automatically identified.

When *esolver_type* is set to `lj` or `dp`, the keyword `pseudo_file` and `pseudo_type` is needless.

Different types of pseudopotentials can be used for different elements, but note that the XC functionals assigned by all pseudopotentials should be the same one. If not, the choice of XC functional must be set explicitly using the *dft_functional* keyword.

Common sources of the pseudopotential files include:

1. Quantum ESPRESSO.

2. SG15-ONCV.
3. DOJO.
4. BLPS.

NUMERICAL_ORBITAL

Numerical atomic orbitals are only needed for `LCAO` calculations. Thus this section will be neglected in calcultions with plane wave basis. In the above example, numerical atomic orbitals is specified for the element `Si`:

```
Si_gga_8au_60Ry_2s2p1d.orb //numerical_orbital_file
```

'Si_gga_8au_60Ry_2s2p1d.orb' is name of the numerical orbital file. Again here the path is not specified, which means that this file is located in the work directory.

Numerical atomic orbitals may be downloaded from the official website.

LATTICE_CONSTANT

The lattice constant of the system in unit of Bohr.

LATTICE_VECTORS

The lattice vectors of the unit cell. It is a 3by3 matrix written in 3 lines. Please note that *the lattice vectors given here are scaled by the lattice constant*. This section must be removed if the type Bravais lattice is specified using the input parameter `latname`. (See *input parameters*.)

LATTICE_PARAMETERS

This section is only relevant when `latname` (see *input parameters*) is used to specify the Bravais lattice type. The example above is a fcc lattice, where no additional information except the lattice constant is required to determine the geometry of the lattice.

However, for other types of Bravais lattice, other parameters might be necessary. In that case, the section `LATTICE_PARAMETERS` must be present. It contains **one single line** with some parameters (separated by blank space if multiple parameters are needed), where the number of parameters required depends on specific type of lattice.

The three lattice vectors v1, v2, v3 (in units of lattice constant) are generated in the following way:

- latname = "sc": the LATTICE_PARAMETERS section is not required:

```
v1 = (1, 0, 0)
v2 = (0, 1, 0)
v3 = (0, 0, 1)
```

- latname = "fcc": the LATTICE_PARAMETERS section is not required:

```
v1 = (-0.5, 0, 0.5)
v2 = (0, 0.5, 0.5)
v3 = (-0.5, 0.5, 0)
```

- latname = "bcc" : the LATTICE_PARAMETERS section is not required:

```
v1 = (0.5, 0.5, 0.5)
v2 = (-0.5, 0.5, 0.5)
v3 = (-0.5, -0.5, 0.5)
```

- latname = "hexagonal" : One single parameter is required in the LATTICE_PARAMETERS section, being the ratio between axis length c/a. Denote it by x then:

```
v1 = (1.0, 0, 0)
v2 = (-0.5, sqrt(3)/2, 0)
v3 = (0, 0, x)
```

- latname = "trigonal" : One single parameter is required in the LATTICE_PARAMETERS section, which specifies cosγ with γ being the angle between any pair of crystallographic vectors. Denote it by x then:

```
v1 = (tx, -ty, tz)
v2 = (0, 2ty, tz)
v3 = (-tx, -ty, tz)
```

  where tx=sqrt((1-x)/2), ty=sqrt((1-x)/6), and tz=sqrt((1+2x)/3).

- latname = "st" (simple tetragonal) : One single parameter is required in the LATTICE_PARAMETERS section, which gives ratio between axis length c/a. Denote it by x then:

```
v1 = (1, 0, 0)
v2 = (0, 1, 0)
v3 = (0, 0, x)
```

- latname = "bct" (body-centered tetragonal) : One single parameter is required in the LATTICE_PARAMETERS section, which gives ratio between axis length c/a. Denote it by x then:

```
v1 = (0.5, -0.5, x)
v2 = (0.5, 0.5, x)
v3 = (-0.5, -0.5, x)
```

- latname = "so" (simple orthorhombic) : Two parameters are required in the LATTICE_PARAMETERS section, which gives ratios between axis length b/a and c/a. Denote them by x, y then:

```
v1 = (1, 0, 0)
v2 = (0, x, 0)
v3 = (0, 0, y)
```

- latname = "baco" (base-centered orthorhombic) : Two parameters are required in the LATTICE_PARAMETERS section, which gives ratios between axis length b/a and c/a. Denote them by x, y then:

```
v1 = (0.5, x/2, 0)
v2 = (-0.5, x/2, 0)
v3 = (0, 0, y)
```

- latname = "fco" (face-centered orthorhombic) : Two parameters are required in the LATTICE_PARAMETERS section, which gives ratios between axis length b/a and c/a. Denote them by x, y then:

```
v1 = (0.5, 0, y/2)
v2 = (0.5, x/2, 0)
v3 = (0, x/2, y/2)
```

- latname = "bco" (body-centered orthorhombic) : Two parameters are required in the LATTICE_PARAMETERS section, which gives ratios between lattice vector length b/a and c/a. Denote them by x, y then:

```
v1 = (0.5, x/2, y/2)
v2 = (-0.5, x/2, y/2)
v3 = (-0.5, -x/2, y/2)
```

- latname = "sm" (simple monoclinic) : Three parameters are required in the LATTICE_PARAMETERS section, which are the ratios of lattice vector length b/a, c/a as well as the cosine of angle between axis a and b. Denote them by x, y, z then:

```
v1 = (1, 0, 0)
v2 = (x*z, x*sqrt(1-z^2, 0)
v3 = (0, 0, y)
```

- latname = "bacm" (base-centered monoclinic) : Three parameters are required in the LATTICE_PARAMETERS section, which are the ratios of lattice vector length b/a, c/a as well as the cosine of angle between axis a and b. Denote them by x, y, z then:

```
v1 = (0.5, 0, -y/2)
v2 = (x*z, x*sqrt(1-z^2), 0)
v3 = (0.5, 0, y/2)
```

- latname = "triclinic" : Five parameters are required in the LATTICE_PARAMETERS section, namely the ratios b/a, c/a; the cosines of angle ab, ac, bc. Denote them by x,y,m,n,l, then:

```
v1 = (1, 0, 0)
v2 = (x*m, x*sqrt(1-m^2), 0)
v3 = (y*n, y*(l-n*m/sqrt(1-m^2)), y*fac)
```

where $fac = \frac{\sqrt{1+2*m*n*l-m^2-n^2-l^2}}{\sqrt{1-m^2}}$

ATOMIC_POSITIONS

This section specifies the positions and other information of individual atoms.

The first line signifies method that atom positions are given, the following options are supported:

- `Direct` : coordinates of atom positions below would in fraction coordinates.
- `Cartesian` : Cartesian coordinates in unit of 'LATTICE_CONSTANT".
- `Cartesian_au` : Cartesian coordinates in unit of Bohr, same as setting of `Cartesian` with `LATTICE_CONSTANT` = 1.0 .
- `Cartesian_angstrom` : Cartesian coordinates in unit of Angstrom, same as setting of `Cartesian` with `LATTICE_CONSTANT` = 1.889726125457828 .
- `Cartesian_angstrom_center_xy` : Cartesian coordinates in unit of Angstrom, with Direct coordinate (0.5, 0.5, 0.0) as reference.
- `Cartesian_angstrom_center_xz` : Cartesian coordinates in unit of Angstrom, with Direct coordinate (0.5, 0.0, 0.5) as reference…
- `Cartesian_angstrom_center_yz` : Cartesian coordinates in unit of Angstrom, with Direct coordinate (0.0, 0.5, 0.5) as reference…
- `Cartesian_angstrom_center_xyz` : Cartesian coordinates in unit of Angstrom, with Direct coordinate (0.5, 0.5, 0.5) as reference…

The following three lines tells the elemental type (`Fe`), the initial magnetic moment (`1.0`), and the number of atoms for this particular element (`2`) repsectively. Notice this magnetic moment will be a default value for every atom of this type but will be overrided if one define it for each atom by keyword(see below).

The last two lines in this example are the coordinates of atomic positions. There are three numbers in each line, which specifies the atomic positions, following by other parameters marked by keywords.

More Key Words

Several other parameters could be defined after the atom position using key words :

- `m` or NO key word: three numbers, which take value in 0 or 1, control how the atom move in geometry relaxation calculations. In example below, the numbers `0 0 0` following the coordinates of the first atom means this atom are *not allowed* to move in all three directions, and the numbers `1 1 1` following the coordinates of the second atom means this atom *can* move in all three directions.
- `v` or `vel` or `velocity`: set the three components of initial velocity of atoms in geometry relaxation calculations(e. g. `v 1.0 1.0 1.0`).
- `mag` or `magmom` : set the start magnetization for each atom. In colinear case only one number should be given. In non-colinear case one have two choice:either set one number for the norm of magnetization here and specify two polar angle later(e. g. see below), or set three number for the xyz commponent of magnetization here (e. g. `mag 0.0 0.0 1.0`). Note that if this parameter is set, the initial magnetic moment setting in the second line will be overrided.
  - `angle1`: in non-colinear case, specify the angle between c-axis and real spin, in angle measure instead of radian measure
  - `angle2`: in non-colinear case, specify angle between a-axis and real spin in projection in ab-plane , in angle measure instead of radian measure

    e.g.:

    ```
    Fe
    1.0
    2
    0.0 0.0 0.0 m 0 0 0 mag 1.0 angle1 90 angle2 0
    0.5 0.5 0.5 m 1 1 1 mag 1.0 angle1 90 angle2 180
    ```

  - Default: If users do not specalize a finite magnetic moment for all atoms in a magnetic calculations (`nspin==2 || nspin == 4`), e.g.:

    ```
    Fe
    0.0
    2
    0.0 0.0 0.0 m 0 0 0
    0.5 0.5 0.5 m 1 1 1

    O
    0.0
    2
    0.0 0.0 0.0 m 0 0 0
    0.5 0.5 0.5 m 1 1 1
    ```

    For `nspin==2`, we will autoset atomic magmon is `1.0`:

    ```
    Fe
    1.0
    ```

(continues on next page)

(continued from previous page)

```
2
0.0 0.0 0.0 m 0 0 0
0.5 0.5 0.5 m 1 1 1

Fe
1.0
2
0.0 0.0 0.0 m 0 0 0
0.5 0.5 0.5 m 1 1 1
```

For `nspin==4`, we will autoset atomic magmon as follow:

```
Fe
0.0
2
0.0 0.0 0.0 m 0 0 0 mag 1 1 1
0.5 0.5 0.5 m 1 1 1 mag 1 1 1

O
0.0
2
0.0 0.0 0.0 m 0 0 0 mag 1 1 1
0.5 0.5 0.5 m 1 1 1 mag 1 1 1
```

However, this autoset will not be vaild once `STRU` specalize a finite magnetic for any single atom.

13.3 The KPT file

- *Generate k-mesh automatically*
- *Set k-points explicitly*
- *Band structure calculations*

ABACUS uses periodic boundary conditions for both crystals and finite systems. For isolated systems, such as atoms, molecules, clusters, etc., one uses the so-called supercell model. Lattice vectors of the supercell are set in the `STRU` file. For the input k-point (`KPT`) file, the file should either contain the k-point coordinates and weights or the mesh size for creating the k-point gird. Both options are allowed in `ABACUS`.

13.3.1 Gamma-only Calculations

In ABACUS, we offer th option of running gamma-only calculations for LCAO basis by setting *gamma_only* to be 1. Due to details of implementation, gamma-only calculation will be slightly faster than running a non gamma-only calculation and explicitly setting gamma point to be the only the k-point, but the results should be consistent.

> If gamma_only is set to 1, the KPT file will be overwritten. So make sure to turn off gamma_only for multi-k calculations.

13.3.2 Generate k-mesh automatically

To generate k-mesh automatically, it requires the input subdivisions of the Brillouin zone in each direction and the origin for the k-mesh. ABACUS uses the Monkhorst-Pack method to generate k-mesh, and the following is an example input k-point (`KPT`) file used in `ABACUS`.

```
K_POINTS //keyword for start
0 //total number of k-point, `0' means generate automatically
Gamma //which kind of Monkhorst-Pack method, `Gamma' or `MP'
2 2 2 0 0 0 //first three number: subdivisions along recpri. vectors
            //last three number: shift of the mesh
```

In the above example, the first line is a keyword, and it can be set as `K_POINTS`, or `KPOINTS` or just `K`. The second line is an integer, and its value determines how to get k-points. In this example, `0` means using Monkhorst-Pack (MP) method to generate k-points automatically.

The third line tells the input type of k-points, `Gamma` or `MP`, different Monkhorst Pack (MP) method. Monkhorst-Pack (MP) is a method which uses the uniform k-points sampling in Brillouin-zone, while `Gamma` means the Γ-centered Monkhorst-Pack method. The first three numbers of the last line are integers, which give the MP k grid dimensions, and the rest three are real numbers, which give the offset of the k grid. In this example, the numbers `0 0 0` means that there is no offset, and this is the a standard 2by2by2 k grid.

back to top

13.3.3 Set k-points explicitly

If the user wants to set up the k-points explicitly, the input k-point file should contain the k-point coordinates and weights. An example is given as follows:

```
K_POINTS //keyword for start
8 //total number of k-point
Direct //`Direct' or `Cartesian' coordinate
0.0 0.0 0.0 0.125 //coordinates and weights
0.5 0.0 0.0 0.125
0.0 0.5 0.0 0.125
0.5 0.5 0.0 0.125
0.0 0.0 0.5 0.125
0.5 0.0 0.5 0.125
0.0 0.5 0.5 0.125
0.5 0.5 0.5 0.125
```

back to top

13.3.4 Band structure calculations

ABACUS uses specified high-symmetry directions of the Brillouin zone for band structure calculations. The third line of k-point file should start with 'Line' or 'Line_Cartesian' for line mode. 'Line' means the positions below are in Direct coordinates, while 'Line_Cartesian' means in Cartesian coordinates:

```
K_POINTS // keyword for start
6 // number of high symmetry lines
Line // line-mode
0.5 0.0 0.5 20 // X
0.0 0.0 0.0 20 // G
0.5 0.5 0.5 20 // L
```

(continues on next page)

(continued from previous page)

```
0.5 0.25 0.75 20 // W
0.375 0.375 0.75 20 // K
0.0 0.0 0.0 1 // G
```

The fourth line and the following are special k-point coordinates and number of k-points between this special k-point and the next.

back to top

CHAPTER

FOURTEEN

HOW TO CITE

The following references are required to be cited when using ABACUS. Specifically:

- **For general purpose:**

  Mohan Chen, G. C. Guo, and Lixin He. "Systematically improvable optimized atomic basis sets for ab initio calculations." Journal of Physics: Condensed Matter 22.44 (2010): 445501.

  Pengfei Li, et al. "Large-scale ab initio simulations based on systematically improvable atomic basis." Computational Materials Science 112 (2016): 503-517.

- **If Stochastic DFT is used:**

  Qianrui Liu, and Mohan Chen. "Plane-Wave-Based Stochastic-Deterministic Density Functional Theory for Extended Systems." https://arxiv.org/abs/2204.05662.

- **If DFT+U is used:**

  Xin Qu, et al. "DFT+ U within the framework of linear combination of numerical atomic orbitals." The Journal of Chemical Physics (2022).

- **If second generation numerical orbital basis is used:**

  Peize Lin, Xinguo Ren, and Lixin He. "Strategy for constructing compact numerical atomic orbital basis sets by incorporating the gradients of reference wavefunctions." Physical Review B 103.23 (2021): 235131.

- **If berry curvature calculation is used in LCAO base:**

  Gan Jin, Daye Zheng, and Lixin He. "Calculation of Berry curvature using non-orthogonal atomic orbitals." Journal of Physics: Condensed Matter 33.32 (2021): 325503.

- **If DeePKS is used:**

  Wenfei Li, Qi Ou, et al. "DeePKS+ABACUS as a Bridge between Expensive Quantum Mechanical Models and Machine Learning Potentials." https://arxiv.org/abs/2206.10093.

- **If hybrid functional is used:**

  Peize Lin, Xinguo Ren, and Lixin He. "Efficient Hybrid Density Functional Calculations for Large Periodic Systems Using Numerical Atomic Orbitals." Journal of Chemical Theory and Computation 2021, 17(1), 222–239.

  Peize Lin, Xinguo Ren, and Lixin He. "Accuracy of Localized Resolution of the Identity in Periodic Hybrid Functional Calculations with Numerical Atomic Orbitals." Journal of Physical Chemistry Letters 2020, 11, 3082-3088.

CHAPTER

FIFTEEN

DEVELOPMENT TEAM

The current development team consists the following research groups/affiliations:

- University of Science and Technology of China (Dr. Lixin He)
- Peking University (Dr. Mohan Chen)
- Institute of Physics, Chinese Academy of Sciences (Dr. Xinguo Ren)
- Beijing AI for Science Institute
- Institute of Artificial Intelligence, Hefei Comprehensive National Science Center.

CHAPTER

SIXTEEN

ABACUS CONTRIBUTION GUIDE

16.1 Contribution Process

We welcome contributions from the open source community. The technical guide is provided in *Contributing to ABACUS*. Here is the basic contribution process:

- **Find out issues to work on.** We assume you already have a good idea on what to do, otherwise the issue tracker and discussion panel provide good starting points to find out what to work on and to get familiar with the project.
- **Approach the issue.** It is suggested to submit new issues before coding out changes to involve more discussions and suggestions from development team. Refer to the technical guide in *Contributing to ABACUS* when needed.
- **Open a pull request.** The ABACUS developers review the pull request (PR) list regularly. If the work is not ready, convert it to draft until finished, then you can mark it as "Ready for review". It is suggested to open a new PR through forking a repo and creating a new branch on you Github account. A new PR should include as much information as possible in `description` when submmited. Unittests or CI tests are required for new PRs.
- **Iterate the pull request.** All pull requests need to be tested through CI before reviewing. A pull request might need to be iterated several times before accepted, so splitting a long PR into parts reduces reviewing difficulty for us.

CHAPTER

SEVENTEEN

CONTRIBUTING TO ABACUS

First of all, thank you for taking time to make contributions to ABACUS! This file provides the more technical guidelines on how to realize it. For more non-technical aspects, please refer to the *ABACUS Contribution Guide*

17.1 Table of Contents

17.2 Got a question?

Please referring to our GitHub issue tracker, and our developers are willing to help. If you find a bug, you can help us by submitting an issue to our GitHub Repository. Even better, you can submit a Pull Request with a patch. You can request a new feature by submitting an issue to our GitHub Repository. If you would like to implement a new feature, please submit an issue with a proposal for your work first, and that ensures your work collaborates with our development road map well. For a major feature, first open an issue and outline your proposal so that it can be discussed. This will also allow us to better coordinate our efforts, prevent duplication of work, and help you to craft the change so that it is successfully accepted into the project.

17.3 Structure of the package

Please refer to *our instructions* on how to installing ABACUS. The source code of ABACUS is based on several modules. Under the ABACUS root directory, there are the following folders:

- `cmake`: relevant files for finding required packages when compiling the code with cmake;
- `docs`: documents and supplementary info about ABACUS;
- `examples`: some examples showing the usage of ABACUS;
- `source`: the source code in separated modules, under which a `test` folder for its unit tests;
- `tests`: End-to-end test cases;
- `tools`: the script for generating the numerical atomic orbitals.

For those who are interested in the source code, the following figure shows the structure of the source code.

```
|-- module_base                 A basic module including
|   |                           (1) Mathematical library interface functions: BLAS,␣
↪LAPACK, Scalapack;
|   |                           (2) Custom data classes: matrix, vector definitions␣
↪and related functions;
|   |                           (3) Parallelization functions: MPI, OpenMP;
|   |                           (4) Utility functions: timer, random number generator,
↪ etc.
|   |                           (5) Global parameters: input parameters, element␣
↪names, mathematical and physical constants.
|   |-- module_container        The container module for storing data and performing␣
↪operations on them and on different architectures.
|-- module_basis                Basis means the basis set to expand the wave function.
|   |-- module_ao               Atomic orbital basis set to be refactored.
|   |-- module_nao              New numerical atomic orbital basis set for two-center␣
↪integrals in LCAO calculations
|   `-- module_pw               Data structures and relevant methods for planewave␣
↪involved calculations
|-- module_cell                 The module for defining the unit cell and its␣
↪operations, and reading pseudopotentials.
|   |-- module_neighbor         The module for finding the neighbors of each atom in␣
↪the unit cell.
|   |-- module_paw              The module for performing PAW calculations.
|   |-- module_symmetry         The module for finding the symmetry operations of the␣
↪unit cell.
|-- module_elecstate            The module for defining the electronic state and its␣
↪operations.
|   |-- module_charge           The module for calculating the charge density, charge␣
↪mixing
|   |-- potentials              The module for calculating the potentials, including␣
↪Hartree, exchange-correlation, local pseudopotential, etc.
|-- module_esolver              The module defining task-specific driver of␣
↪corresponding workflow for evaluating energies, forces, etc., including lj, dp, ks,␣
↪sdft, ofdft, etc.
|   |                           TDDFT, Orbital-free DFT, etc.
|-- module_hamilt_general       The module for defining general Hamiltonian that can␣
↪be used both in PW and LCAO calculations.
|   |-- module_ewald            The module for calculating the Ewald summation.
|   |-- module_surchem          The module for calculating the surface charge␣
↪correction.
|   |-- module_vdw              The module for calculating the van der Waals␣
```

(continues on next page)

(continued from previous page)

```
↪correction.
|   |-- module_xc               The module for calculating the exchange-correlation␣
↪energy and potential.
|-- module_hamilt_lcao          The module for defining the Hamiltonian in LCAO␣
↪calculations.
|   |-- hamilt_lcaodft          The module for defining the Hamiltonian in LCAO-DFT␣
↪calculations.
|   |   |-- operator_lcao       The module for defining the operators in LCAO-DFT␣
↪calculations.
|   |-- module_deepks           The module for defining the Hamiltonian in DeepKS␣
↪calculations.
|   |-- module_dftu             The module for defining the Hamiltonian in DFT+U␣
↪calculations.
|   |-- module_gint             The module for performing grid integral in LCAO␣
↪calculations.
|   |-- module_hcontainer       The module for storing the Hamiltonian matrix in LCAO␣
↪calculations.
|   `-- module_tddft            The module for defining the Hamiltonian in TDDFT␣
↪calculations.
|-- module_hamilt_pw            The module for defining the Hamiltonian in PW␣
↪calculations.
|   |-- hamilt_ofdft            The module for defining the Hamiltonian in OFDFT␣
↪calculations.
|   |-- hamilt_pwdft            The module for defining the Hamiltonian in PW-DFT␣
↪calculations.
|   |   |-- operator_pw         The module for defining the operators in PW-DFT␣
↪calculations.
|   `-- hamilt_stodft           The module for defining the Hamiltonian in STODFT␣
↪calculations.
|-- module_hsolver              The module for solving the Hamiltonian with different␣
↪diagonalization methods, including CG, Davidson in PW
|   |                           calculations, and scalapack and genelpa in LCAO␣
↪calculations.
|-- module_io                   The module for reading of INPUT files and output␣
↪properties including band structure, density of states, charge density, etc.
|-- module_md                   The module for performing molecular dynamics.
|-- module_psi                  The module for defining the wave function and its␣
↪operations.
|-- module_relax                The module for performing structural optimization.
|   |-- relax_new               The module for performing structural optimization␣
↪with new algorithm, optimized for cell and ion simultaneously.
|   `-- relax_old               The module for performing structural optimization␣
↪with old algorithm, optimized for cell and ion separately.
|-- module_ri                   The module for performing RI calculations.
```

17.4 Submitting an Issue

Before you submit an issue, please search the issue tracker, and maybe your problem has been discussed and fixed. You can submit new issues by filling our issue forms. To help us reproduce and confirm a bug, please provide a test case and building environment in your issue.

17.5 Comment style for documentation

ABACUS uses Doxygen to generate docs directly from `.h` and `.cpp` code files.

For comments that need to be shown in documents, these formats should be used – **Javadoc style** (as follow) is recommended, though Qt style is also ok. See it in official manual.

A helpful VS Code extension – Doxygen Documentation Generator, can help you formating comments.

An practical example is class LCAO_Deepks, the effects can be seen on readthedocs page

- Tips
  - Only comments in .h file will be visible in generated by Doxygen + Sphinx;
  - Private class members will not be documented;
  - Use Markdown features, such as using a empty new line for a new paragraph.
- Detailed Comment Block

```
/**
* ... text ...
*/
```

- Brief + Detailed Comment Block

```
/// Brief description which ends at this dot. Details follow
/// here.

/// Brief description.
/** Detailed description. */
```

- Comments After the Item: Add a "<"

```
int var; /**<Detailed description after the member */
int var; ///<Brief description after the member
```

- Parameters usage: `[in],[out],[in,out] description` *e.g.*

```
void foo(int v/**< [in] docs for input parameter v.*/);
```

  or use `@param` command.

- Formula
  - inline: `\f$myformula\f$`
  - separate line: `\f[myformula\f]`
  - environment: `\f{environment}{myformula}`
  - *e.g.*

```
\f{eqnarray*}{
        g &=& \frac{Gm_2}{r^2} \\
        &=& \frac{(6.673 \times 10^{-11}\,\mbox{m}^3\,\mbox{kg}^{-1}\,
            \mbox{s}^{-2})(5.9736 \times 10^{24}\,\mbox{kg})}{(6371.01\,\mbox
↪{km})^2} \\
        &=& 9.82066032\,\mbox{m/s}^2
\f}
```

17.6 Code formatting style

We use `clang-format` as our code formatter. The `.clang-format` file in root directory describes the rules to conform with. For Visual Studio Code developers, the official extension of C/C++ provided by Microsoft can help you format your codes following the rules. With this extension installed, format your code with `shift+command/alt+f`. Configure your VS Code settings as `"C_Cpp.clang_format_style": "file"` (you can look up this option by pasting it into the search box of VS Code settings page), and all this stuff will take into effect. You may also set `"editor.formatOnSave": true` to avoid formatting files everytime manually.

17.7 Adding a unit test

We use GoogleTest as our test framework. Write your test under the corresponding module folder at `abacus-develop/tests`, then append the test to `tests/CMakeLists.txt`. If there are currently no unit tests provided for the module, do as follows. `module_base` provides a simple demonstration.

- Add a folder named `test` under the module.
- Append the content below to `CMakeLists.txt` of the module:

```
IF (BUILD_TESTING)
  add_subdirectory(test)
endif()
```

- Add a blank `CMakeLists.txt` under `module*/test`.

To add a unit test:

- Write your test under `GoogleTest` framework.
- Add your testing source code with suffix `*_test.cpp` in `test` directory.
- Append the content below to `CMakeLists.txt` of the module:

```
AddTest(
  TARGET <module_name>_<test_name> # this is the executable file name of the test
  SOURCES <test_name>.cpp

  # OPTIONAL: if this test requires external libraries, add them with "LIBS"␣
↪statement.
  LIBS math_libs # `math_libs` includes all math libraries in ABACUS.
)
```

- Build with `-D BUILD_TESTING=1` flag, `cmake` will look for `GoogleTest` in the default path (usually `/usr/local`); if not found, you can specify the path with `-D GTEST_DIR`. You can find built testing programs under `build/source/<module_name>/test`.
- Follow the installing procedure of CMake. The tests will move to `build/test`.

- Considering `-D BUILD_TESTING=1`, the compilation will be slower compared with the case `-D BUILD_TESTING=0`.

17.8 Running unit tests

1. Compiling ABACUS with unit tests.

   In order to run unit tests, ABACUS needs to be configured with `-D BUILD_TESTING=ON` flag. For example:

   ```
   cmake -B build -DBUILD_TESTING=ON
   ```

   then build ABACUS and unit testing with

   ```
   cmake --build build -j${number of processors}
   ```

   It is import to run the folloing command before running unit tests:

   ```
   cmake --install build
   ```

   to install mandatory supporting input files for unit tests. If you modified the unit tests to add new tests or learn how to write unit tests, it is convenient to run

   ```
   cmake --build build -j${number of processors} --target ${unit test name}
   ```

   to build a specific unit test. And please remember to run `cmake --install build` after building the unit test if the unit test requires supporting input files.

2. Running unit tests

   The test cases are located in `build/source/${module_name}/test` directory. Note that there are other directory names for unit tests, for example, `test_parallel` for running parallel unit tests, `test_pw` for running unit tests only used in plane wave basis calculation.

   You can run a single test in the specific directory. For example, run

   ```
   ./cell_unitcell_test
   ```

   under the directory of `build/source/module_cell/test` to run the test `cell_unitcell_test`. However, it is more convenient to run unit tests with `ctest` command under the `build` directory. You can check all unit tests by

   ```
   ctest -N
   ```

   The results will be shown as

   ```
   Test project /root/abacus/build
   Test   #1: integrated_test
   Test   #2: Container_UTs
   Test   #3: base_blas_connector
   Test   #4: base_blacs_connector
   Test   #5: base_timer
   ...
   ```

   Note that the first one is integrated test, which is not a unit test. It is the test suite for testing the whole ABACUS package. The examples are located in the `tests/integrate` directory.

   To run a subset of tests, run the following command

```
ctest -R <test-match-pattern> -V
```

For example, `ctest -R cell` will perform tests with name matched by `cell`. You can also run a single test with

```
ctest -R <test-name>
```

For example, `ctest -R cell_unitcell_test_readpp` will perform test `cell_unitcell_test_readpp`. To run all the unit tests, together with the integrated test, run

```
cmake --build build --target test ARGS="-V --timeout 21600"
```

in the `abacus-develop` directory.

17.9 Debugging the codes

For the unexpected results when developing ABACUS, GDB will come in handy.

1. Compile ABACUS with debug mode.

   ```
   cmake -B build -DCMAKE_BUILD_TYPE=Debug
   ```

2. After building and installing the executable, enter the input directory, and launch the debug session with `gdb abacus`. For debugging in Visual Studio Code, please set cwd to the input directory, and program to the path of ABACUS executable.

3. Set breakpoints, and run ABACUS by typing "run" in GDB command line interface. If the program hits the breakpoints or exception is throwed, GDB will stop at the erroneous code line. Type "where" to show the stack backtrace, and "print i" to get the value of variable i.

4. For debugging ABACUS in multiprocessing situation, `mpirun -n 1 gdb abacus : -n 3 abacus` will attach GDB to the master process, and launch 3 other MPI processes.

For segmentation faults, ABACUS can be built with Address Sanitizer to locate the bugs.

```
cmake -B build -DENABLE_ASAN=1
```

Run ABACUS as usual, and it will automatically detect the buffer overflow problems and memory leaks. It is also possible to use GDB with binaries built by Address Sanitizer.

Valgrind is another option for performing dynamic analysis.

17.10 Adding a new building component

ABACUS uses CMake as its default building system. To add a new building component:

1. Add an `OPTION` to toggle the component to the `CMakeLists.txt` file under root directory. For example:

   ```
   OPTION(ENABLE_NEW_COMPONENT "Enable new component" OFF)
   ```

2. Add the new component. For example:

```
IF (ENABLE_NEW_COMPONENT)
  add_subdirectory(module_my_new_feature) # if the feature is implemented in a␣
↪subdirectory
  find_package(NewComponent REQUIRED) # if third-party libs are required
  target_link_libraries(${ABACUS_BIN_NAME} PRIVATE NewComponent) # if the␣
↪component is linked
  include_directories(${NewComponent_INCLUDE_DIRS}) # if the component is included
endif()
```

3. Add the required third-party libraries to Dockerfiles.
4. After the changes above are merged, submit another PR to build and test the new component in the CI pipeline.
   - For integration test and unit test: add `-DENABLE_NEW_COMPONENT=ON` to the building step at `.github/workflows/test.yml`.
   - For building test: add `-DENABLE_NEW_COMPONENT=ON` as a new configuration at `.github/workflows/build_test_cmake.yml`.

17.11 Generating code coverage report

This feature requires using GCC compiler. We use `gcov` and `lcov` to generate code coverage report.

1. Add `-DENABLE_COVERAGE=ON` for CMake configure command.

```
cmake -B build -DBUILD_TESTING=ON -DENABLE_COVERAGE=ON
```

2. Build, install ABACUS, and run test cases. Please note that since all optimizations are disabled to gather running status line by line, the performance is drastically decreased. Set a longer time out to ensure all tests are executed.

```
cmake --build build --target test ARGS="-V --timeout 21600"
```

   If configuration fails unfortunately, you can find required files (including three *.cmake and llvm-cov-wrapper), and copy these four files into `/abacus-develop/cmake`. Alternatively, you can define the path with option `-D CMAKE_CURRENT_SOURCE_DIR`.

3. Generate HTML report.

```
cd build/
make lcov
```

Now you can copy `build/lcov` to your local device, and view `build/lcov/html/all_targets/index.html`.

We use Codecov to host and visualize our **code coverage report**. Analysis is scheduled after a new version releases; this action can also be manually triggered.

17.12 Submitting a Pull Request

1. Fork the ABACUS repository. If you already had an existing fork, sync the fork to keep your modification up-to-date.
2. Pull your forked repository, create a new git branch, and make your changes in it:

```
git checkout -b my-fix-branch
```

3. Coding your patch, including appropriate test cases and docs. To run a subset of unit test, use `ctest -R <test-match-pattern>` to perform tests with name matched by given pattern.
4. After tests passed, commit your changes *with a proper message*.
5. Push your branch to GitHub:

```
git push origin my-fix-branch
```

6. In GitHub, send a pull request (PR) with `deepmodeling/abacus-develop:develop` as the base repository. It is required to document your PR following *our guidelines*.
7. After your pull request is merged, you can safely delete your branch and sync the changes from the main (upstream) repository:

- Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

```
git push origin --delete my-fix-branch
```

- Check out the master branch:

```
git checkout develop -f
```

- Delete the local branch:

```
git branch -D my-fix-branch
```

- Update your master with the latest upstream version:

```
git pull --ff upstream develop
```

17.13 Commit message guidelines

A well-formatted commit message leads a more readable history when we look through some changes, and helps us generate change log. We follow up The Conventional Commits specification for commit message format. This format is also required for PR title and message. The commit message should be structured as follows:

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

- Header
  - type: The general intention of this commit
    - `Feature`: A new feature

    * `Fix`: A bug fix
    * `Docs`: Only documentation changes
    * `Style`: Changes that do not affect the meaning of the code
    * `Refactor`: A code change that neither fixes a bug nor adds a feature
    * `Perf`: A code change that improves performance
    * `Test`: Adding missing tests or correcting existing tests
    * `Build`: Changes that affect the build system or external dependencies
    * `CI`: Changes to our CI configuration files and scripts
    * `Revert`: Reverting commits
  - scope: optional, could be the module which this commit changes; for example, `orbital`
  - description: A short summary of the code changes: tell others what you did in one sentence.
- Body: optional, providing detailed, additional, or contextual information about the code changes, e.g. the motivation of this commit, referenced materials, the coding implementation, and so on.
- Footer: optional, reference GitHub issues or PRs that this commit closes or is related to. Use a keyword to close an issue, e.g. "Fix #753".

Here is an example:

```
Fix(lcao): use correct scalapack interface.

`pzgemv_` and `pzgemm_` used `double*` for alpha and beta parameters but not␣
↪`complex*` , this would cause error in GNU compiler.

Fix #753.
```

CHAPTER

EIGHTEEN

LISTS OF CONTINUOUS INTEGRATION (CI) PIPELINES

The directory `./github/workflows` contains the continuous integration (CI) pipelines for the project. The pipelines are written in YAML format and are executed by GitHub Actions. Check the Actions page of the repo for the status of the pipelines.

The tests without mentioning are not executed.

18.1 On Pull Request (PR)

The following CI pipelines are triggered on pull request (PR) creation or update (the user pushes a new commit to the incoming branch):

- Integration test and unit tests (`test.yml`): This pipeline builds ABACUS with all available features, runs integration tests, and runs unit tests.
- Building tests with CMake (`build_test_cmake.yml`): This pipeline builds ABACUS with each feature separately, ensuring: (i) there are no conflicts between features, (ii) the compilation is successful whether the feature is enabled, and (iii) it works well on multiple platforms, i.e. with GNU+OpenBLAS toolchain and Intel+MKL toolchain.
- Rerender the docs site: This pipeline rerenders the documentation site on Read the Docs. It is automatically triggered when the documentation is updated.
- Testing GPU features (`cuda.yml`): This pipeline builds ABACUS with GPU support and runs several tests on the GPU. **Currently disabled for the lack of GPU resource.**

After the PR merges into the main branch, the following pipelines are triggered:

- Building Docker images(`devcontainer.yml`): This pipeline builds the Docker images with latest codes and executables. The images are tagged as `abacus-gnu:latest`, `abacus-intel:latest`, and `abacus-cuda:latest`, and then pushed to the GitHub Container Registry (ghcr.io/deepmodeling) and AliCloud Container mirror(registry.dp.tech/deepmodeling). For example: `docker pull ghcr.io/deepmodeling/abacus-intel:latest`.

18.2 On Routine

- Dynamic analysis (`dynamic_analysis.yml`): This pipeline runs integration tests with AddressSanitizer to detect memory errors. The pipeline is scheduled to run **every Sunday**. The results are published on GitHub Pages.

18.3 On Release

- Coverage test (`coverage.yml`): This pipeline builds ABACUS with all available features, runs integration tests, and runs unit tests. It also measures the code coverage of the tests. The results are published at codecov.io.
- Building tagged Docker images (`image.yml`): The built image is tagged in the pattern of `abacus:3.5.0`, and pushed to the GitHub Container Registry (ghcr.io/deepmodeling) and AliCloud Container mirror(registry.dp.tech/deepmodeling). Use `abacus:latest` to fetch the latest image. For example: `docker pull ghcr.io/deepmodeling/abacus:latest`.

CHAPTER

NINETEEN

FREQUENTLY ASKED QUESTIONS

19.1 General Questions

1. What are the merits of ABACUS in respect of functionality, performance, and/or accuracy?

Users are referred to the introduction of features of ABACUS in the feature list.

19.2 Installation

19.3 Setting up jobs

1. Why pseudopotential files must be provided in LCAO calculation?

The pseudopotentials are used to approximate the potential of nuclear and core electrons, while the numerical orbitals are basis sets used to expand the Hamiltonian. So both pseudopotential and numerical orbital files are needed in LCAO calculation.

2. What is the correlation between pseudopotential and numerical orbital files?

The numerical orbital files are generated for a specific pseudopotential. So the right numerical orbital files must be chosen for a specific pseudopotential. We suggest users choose numerical orbital files and the corresponding pseudopotential files from the ABACUS website because their accuracy has been tested. However, interesting users may also generate their own numerical orbitals for a specific type of pseudopential by using the tools provided in the abacus-develop/tools directory.

3. How to set `ecutwfc` in LCAO calculation? Must it be 100 Ry for a numerical orbital file like `Cu_lda_7.0au_100Ry_2s2p2d`?

It is recommended to set `ecutwfc` to the value that the numerical orbital file suggests, but it is not a must. The `ecutwfc` value only affects the number of FFT grids.

4. Does ABACUS support LCAO calculations accounting for external electric field effects?

Yes, users are referred to documentation on *external electric field*.

5. Can ABACUS calculate non-periodic systems, such as ionic liquids?

Non-periodic systems such as liquid systems can be calculated by using supercell and gamma-only calculation.

6. How to perform spin-orbital coupling (SOC) calculations in ABACUS?

Apart from setting relavant keys (`lspinorb` to 1) in the `INPUT` file, SOC calculations can only be performed with fully-relativistic pseudopotentials. Users are suggested to download fully-relativistic versions of SG15_ONCV pseudopotential files from a website. The numerical orbital files generated from the corresponding scalar-relativistic pseudoptential files by ABACUS (here) can be used in collaboration with the fully-relativistic pseudopotentials.

7. How to restart jobs in abacus?

For restarting SCF calculations, users are referred to the documentation about *continuation of job*. For restarting MD calculations, please see *md_restart*.

8. Can DeePKS model be used for structural optimization calculation? What parameters need to be modified or called?

If you train the DeePKS model with force labels, then the DeePKS model can provide force calculation with the same accuracy as your target method, and can thus be used for structural optimization. To do that, you just need to train the model with force label enabled.

9. How to estimate the max memory consumption?

Run `/usr/bin/time -v mpirun -n 4 abacus`, and locate "Maximum resident set size" in the output log at the end. Please note that this value is the peak memory size of the main MPI process.

10. Why there are two sigma (smearing_sigma and dos_sigma) in some examples for dos calculation?

The tag `smearing_sigma` is used for SCF calculation, and does not affect NSCF calculation. The tag `dos_smearing` is only used for plotting density of states, which does affect SCF or NSCF results. So `smearing_sigma` should not be set in dos calculation.

11. How to set `nbands` and `ncpus`?

For both pw and LCAO calculations, the default value for `nbands` can be found here. Note that the number of CPUs called for a parallel job (i.e., the number specified after the command `mpirun -n`) should be smaller than `nbands`, otherwise the job will fail with an error message `nbands < ncpus`. Note also that for LCAO calculations, `nbands` should always be smaller than `nlocal`, i.e., the number of the atomic orbital basis of the system.

back to top

19.4 Failed jobs

1. Why my calculation is pend by using mpirun?

This is usually caused by overloading of CPUs' memory without specifying thread numbers. ABACUS detects available hardware threads, and provides these information at the beginning of the program if used threads mismatches with hardware availability. User should keep total used threads(i.e. the number of OpenMP threads times the number of MPI processes) no more than the count of available CPUs(can be determined by `lscpu`). Setting `export OMP_NUM_THREADS=1` will solve this problem, or one can use the command like `OMP_NUM_THREADS=1 mpirun -n 8 abacus` to rerun failed jobs.

2. My relaxation failed. How to deal with it?

This is usually caused by the difficulty in converging charge density. Reducing charge mixing coefficient (`mixing_beta`) might help. For large systems up to hundreds of atoms, it is suggested to choose the Kerker mixing method by setting parameter "mixing_gg0" as "1.0".

Sometimes, loose convergence threshold of charge density (parameter "scf_thr") will cause atomic forces not correctly enough, please set it at most "1e-7" for relaxation calculation.

3. Why the program is halted?

If the program prompt something like "KILLED BY SIGNAL: 9 (Killed)", it may be caused by insufficient memory. You can use `dmesg` to print out system info regarding memory management, and check if there is "Out of memory: Killed" at the end of output info. Please try using less processes and threads for calculation, or modify the input parameters requiring less memory.

If the error message is "Segmentation fault", or there is no enough information on the error, please feel free to submit an issue.

19.5 Miscellaneous

1. How to visualize charge density file?

The output file SPIN1_CHG.cube can be visualized by using VESTA.

2. How to change cif file directly to STRU file?

One way to change from cif to STRU is to use the ASE-ABACUS interface. An example of the converting script is provided below:

```
from ase.io import read, write
from pathlib import Path

cs_dir = './'
cs_cif = Path(cs_dir, 'SiO.cif')
cs_atoms = read(cs_cif, format='cif')
cs_stru = Path(cs_dir, 'STRU')
pp = {'Si':'Si.upf','O':'O.upf'}
basis = {'Si':'Si.orb','O':'O.orb'}
write(cs_stru, cs_atoms, format='abacus', pp=pp, basis=basis)
```

3. What is the convergence criterion for the SCF process in ABACUS?

ABACUS applies the density difference between two SCF steps (labeled as `DRHO` in the screen output) as the convergence criterion, which is considered as a more robust choice compared with the energy difference. `DRHO` is calculated via `DRHO = |rho(G)-rho_previous(G)|^2`. Note that the energy difference between two SCF steps (labed as `EDIFF`) is also printed out in the screen output.

**4. Why EDIFF is much slower than DRHO?

For metaGGA calculations, it is normal because in addition to charge density, kinetic density also needs to be considered in metaGGA calculations. In this case, you can try set `mixing_tau = true`. If you find EDIFF is much slower than DRHO for non-metaGGA calculations, please start a new issue to us.

back to top