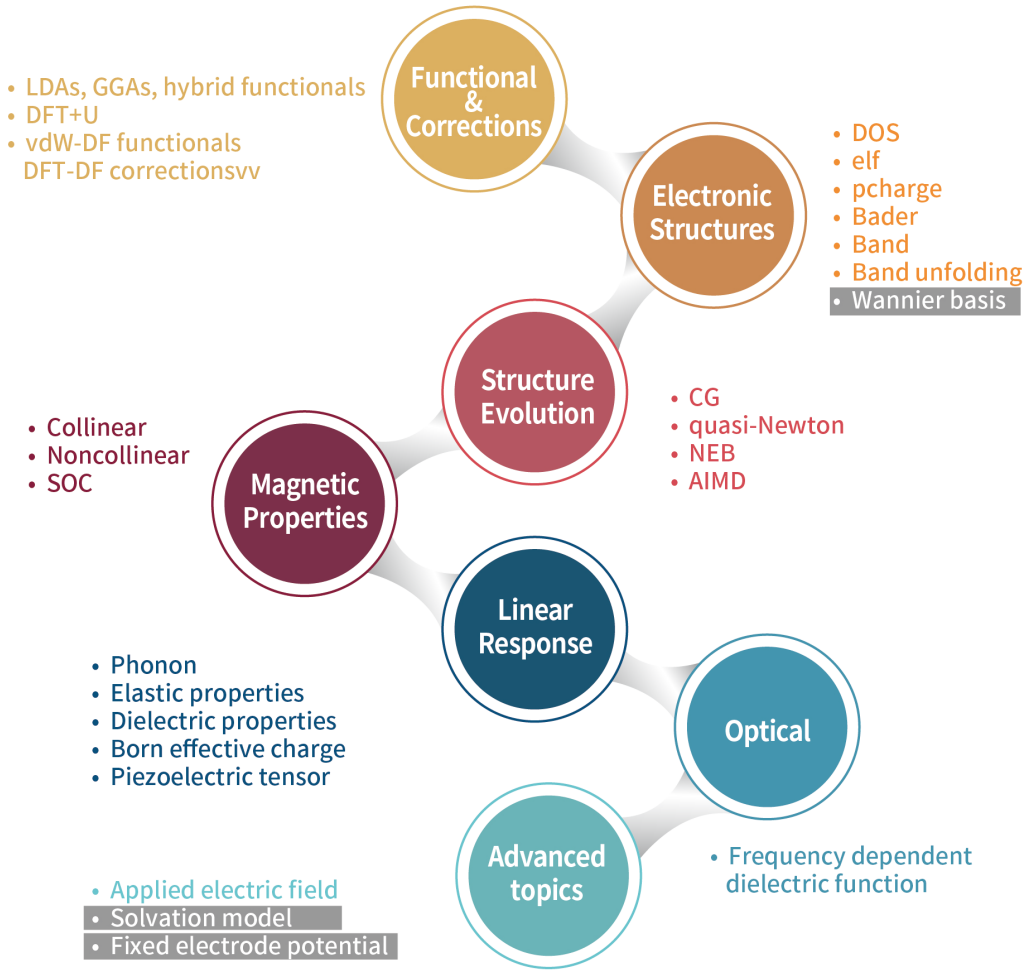
# DS-PAW Manual

2023 HZWtech

2025-09-26

Table of Contents:




The functions highlighted in gray will be supported from DS-PAW 2023A and later versions.

# 1

# Software Introduction

DS-PAW is a first-principles density functional computation program within the Device Studio platform, using plane waves as the basis set and employing the projected augmented wave (PAW) method to construct pseudopotentials. This program can be widely applied in the field of materials science to conduct computational research on materials such as metals, semiconductors, insulators, surfaces, magnetic, non-magnetic, and lithium-ion battery materials; Can accurately predict the electronic distribution of materials; capable of performing calculations such as atomic geometric structure optimization and other functions. This program is stable in performance and has undergone internal testing on millions of cases on Intel and domestic Huawei chips, including various functions and parallel efficiency.

## 1.1 Command Description

### 1.1.1 List command list

- *-lic*
- *-info*
- *-example*
- *-ipp*
- *-mpi*
- *-mpiargs*
- *-pob*

---

### 1.1.2 detail command description

***Command Name:** *-lic*

***Usage:** *-lic* is used to generate a serial number. Execute the command *DS-PAW -lic* in the DS-PAW installation directory to obtain the LicenseNumber.txt file, which is used for license application.

---

***Command Name:** *-info*

***Usage:** *-info* is used to view software copyright information, execute command: *DS-PAW -info*

---

***Command Name:** *-example*

***Usage:** *-example* is used to quickly perform a single calculation, which can check if DS-PAW is installed correctly. Execute the command: *DS-PAW -example*

---

***Command Name:** *-ipp*

***Usage:** *-ipp* is used to view the DS-PAW pseudopotential data information, including the cutoff energy, valence electron number, etc. Execute the command: *DS-PAW -ipp*

---

***Command Name:** *-mpi xxx*

***Usage:** *-mpi* is used to specify the location of the mpi execution program, such as: *-mpi mpirun*

---

***Command Name:** *-mpiargs xxx*

***Usage:** *-mpiargs* is used to specify MPI runtime arguments, such as *-mpiargs -np 16*

---

***Command Name:** *-pob*

***Usage:** *-pob* is used to reasonably allocate the number of cores for parallel computing to speed up the run, short for *parallel over band*, and can be added to the submission command. DS-PAW cannot enable pob in some functional calculations, and will issue a warning and turn off pob in this case

---

## 1.2 run program running

### 1.2.1 submit command to submit execution

Set environment variables:

```
export PATH={DS-PAW INSTALLPATH}/bin:$PATH
```

Serial execution:

```
DS-PAW input.in
```

Parallel execution:

```
DS-PAW -mpi mpirun -mpiargs "-np 16" input.in -pob
```

### 1.2.2 Submit the script to run

If using a queuing system (such as PBS, Slurm, etc.) to submit tasks, as long as the corresponding `.pbs` or `.slurm` scripts are configured, you can submit tasks using *qsub xx.pbs* or *sbatch xx.slurm*.

# 2

# Quickstart

This chapter will introduce the basic usage of various functions of DS-PAW, including: **Structure Relaxation Calculation**, **Self-Consistent Calculation**, **Band (Projected Band) Calculation**, **Density of States (Projected Density of States) Calculation**, **Potential Function Calculation**, **Electron Localization Density Calculation**, **Partial Charge Density Calculation**, **Hybrid Functional Calculation**, **Van der Waals Correction Calculation**, **Dipole Correction Calculation**, **DFT+U Calculation**, **Background Charge Calculation**, **Optical Properties Calculation**, **Frequency Calculation**, **Elastic Constants Calculation**, **Transition State Calculation**, **Phonon Spectrum Calculation**, **Spin-Orbit Coupling Calculation**, **Molecular Dynamics Simulation**, **External Electric Field Calculation**, **Ferroelectric Calculation**, **Bader Charge Analysis**, **Band Unfolding Calculation**, **Dielectric Constant Calculation**, **Piezoelectric Tensor Calculation**, **Fixed Basis Relaxation Calculation**, **Phonon Thermodynamic Properties Calculation**, **Solid State NEB Calculation**, **Solvation Energy Calculation**, **Fixed Potential Calculation**, **Wannier Interpolated Band Calculation**; The parameters of the DS-PAW software can be roughly classified into the following categories: parameters related to the physical structure, parameters related to the calculated properties, parameters related to the calculation accuracy, and parameters related to convergence. Most basic parameters have default values. This chapter introduces a selection of parameters. For the complete parameter list and details, please refer to *Parameters Explanation*.

## 2.1 relax structure calculation

In Density Functional Theory (DFT), structural relaxation refers to changing the initial structures cell and atomic positions to optimize and obtain a local minimum of the total energy. By performing structural relaxation calculations, the forces on each atom can be reduced, leading to a more stable structure (to some extent, the stability of the structure can be verified by calculating the phonon spectrum or frequencies). In general, structures built using modeling software often have large atomic forces. Moreover, even structures optimized by other DFT software may not necessarily have the minimum atomic forces in a different DFT calculation software. Therefore, a structural relaxation calculation is necessary before calculating the specific properties of a structure.

### 2.1.1 $Si$ atom structure relaxation input file

The input file contains the parameter file *relax.in* and the structure file *structure.as*, with *relax.in* as follows:

```
# task type
task = relax
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5

#relax related
relax.max =  60
relax.freedom = atom
relax.convergenceType = force
relax.convergence =  0.05
relax.methods = CG

io.wave = false
io.charge = false
```

The *relax.in* file can be roughly divided into four sections of parameters:

**The first part specifies the calculation type, controlled by the task parameter:**

- – `task`: Specifies the calculation type. This calculation is for relaxation, i.e., structure relaxation.

**The second part specifies system-related parameters, which start with sys., and generally relate to the systems structure, functional, magnetism, and symmetry:**

- `sys.structure`: Specifies the structure file of the system. DS-PAW supports structure file formats of *.as* and *.h5* (early JSON files are supported but users are not recommended to use them, and subsequent DS-PAW releases will completely discard the JSON format output). The *.as* file can be generated directly using the Device Studio software or constructed manually.
- `sys.symmetry`: Sets whether to use symmetry during the DS-PAW calculation;
- `sys.functional`: Sets the functional, currently supporting **LDA**, **PBE**, and various modified functionals;
- `sys.spin` : Sets the magnetism of the system. Since **Si** is non-magnetic, set sys.spin to **none**;

**Part three specifies parameters related to the calculation, which are prefixed with cal.:**

- `cal.methods`: Sets the self-consistent electronic step optimization method, 2 indicates the **Residual minimization** method is used;
- `cal.smearing`: Specifies the partial occupation method for each wave function, with 1 indicating the use of **Gaussian smearing**.
- `cal.ksamping`: Method for automatically generating the Brillouin zone **k-point** grid, G represents using the **Gamma centered** method.
- `cal.kpoints`: Set the sampling size of the Brillouin zone **k-point** grid. The size of the K-point grid generally needs to be set according to the size of the systems lattice and its periodicity.

**Part Four specifies parameters related to structure relaxation, such as the relaxation method, relaxation type, and relaxation accuracy. Structure relaxation refers to optimizing atomic positions to obtain a structure**

**with a local minimum total energy, also commonly known as ionic step optimization;**

- `relax.max`: Sets the maximum number of ionic steps for structural relaxation;
- `relax.freedom`: Sets the degrees of freedom for structural relaxation. `atom` means only relax atomic positions; `volume` means only relax lattice volume; `all` means relax atomic positions, cell shape, and volume;
- `relax.convergenceType`: Sets the criteria type for structural relaxation convergence, with force indicating that atomic forces are used as the criterion, and energy as another optional value;
- `relax.convergence`: Sets the convergence accuracy of atomic forces during structural relaxation.
- `relax.methods` : Sets the method for structural relaxation, CG represents the conjugate gradient method;

The *structure.as* file is referenced as follows:

```
Total number of atoms
2
Lattice
0.00 2.75 2.75
2.75 0.00 2.75
2.75 2.75 0.00
Direct
Si -0.115000000 -0.125000000 -0.125000000
Si 0.125000000 0.125000000 0.125000000
```

The structure of the *structure.as* file is fixed, and the corresponding information must be written precisely line by line.

- The first line is a fixed prompt line.
- The second line is the total number of atoms.
- This line is a fixed prompt line
- Lines four to six contain the unit cell information.
- The seventh line specifies the format of atomic coordinates, with options **Direct** and **Cartesian**.
- Atomic coordinate information starts from the eighth line, and each line must begin with the name of the atom whose coordinates are described.

To demonstrate the structural changes before and after relaxation, this example manually changes the x-coordinate of the first Si atom from **-0.125** to **-0.115**.

> **Note**
>
> 1. **To fix atoms, add the Fix_x Fix_y Fix_z tag on line 7, and then add F or T at the corresponding positions for each atom, where F means not fixed and T means fixed.**

```
Direct Fix_x Fix_y Fix_z
Si -0.115000000 -0.125000000 -0.125000000 F F F
Si 0.125000000 0.125000000 0.125000000 T T T
```

### 2.1.2 run program running

After preparing the input files, upload the files *relax.in* and *structure.as* to the environment where DS-PAW is installed. This section will use the Linux environment as an example.

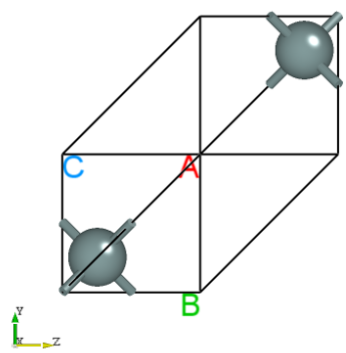
Running the software in a Linux environment without a graphical interface differs significantly from running programs in Windows. In Linux, you need to execute programs through the command line. Generally, you need to load the environment variables first. Usually, the necessary environment variables are written to a text file or **~/.bashrc**, and the environment is loaded using the **source** command. After the environment is loaded, run *DS-PAW relax.in* for single-machine calculations. For parallel computing, run *DS-PAW -mpi mpirun -mpiargs -n 2 relax.in*. **-mpi** specifies the name of mpirun. **-mpiargs** specifies the arguments following mpirun. See Section *Software Introduction* for a command introduction. To submit jobs using queuing systems (e.g., PBS, slurm), configure the corresponding *.pbs* or *.slurm* script first, then submit the job using *qsub DS-PAW.pbs* or *sbatch DS-PAW.slurm*.

### 2.1.3 Analysis Results Analysis

Based on the input files mentioned above, after computation, the following output files will be generated: *DS-PAW.log*, *relax.h5*, and *latestStructure.as*.

- *DS-PAW.log*: The log file generated after the DS-PAW calculation;
- *relax.h5* : The h5 output file corresponding to the relaxation calculation. See section *Output File Format Specification* for structural analysis. This h5 file can be read by DS-PAW for continued calculation;
- *latestStructure.as*: The final structure file in .as format after relaxation, allowing for direct data viewing;

Drag *latestStructure.as* into Device Studio to view the structure as shown below:




The unit cell information after relaxation can be found in the *latestStructure.as* file:

```
Total number of atoms
2
Lattice
0.0000000000000000    2.7500000000000000    2.7500000000000000
2.7500000000000000    0.0000000000000000    2.7500000000000000
2.7500000000000000    2.7500000000000000    0.0000000000000000
Direct
Si  0.8801735223171917  0.8748246492235915  0.8748246492235915
Si  0.1298264776828063  0.1251753507764085  0.1251753507764085
```

This structural relaxation calculation performed 3 ionic steps. In the final relaxed configuration, the x-coordinate of the manually moved Si atom was corrected.

> **Note**
>
> 1. **The single-machine DS-PAW execution command is the software name + input file name. If your input file name is abc.in, simply execute DS-PAW abc.in.**
> 2. **The convergence criterion for this relaxation calculation is chosen as atomic force. If energy is to be used as the convergence criterion, you can set relax.convergenceType = energy.**

## 2.2 SCF Self-Consistent Calculation

Self-consistent calculations yield the charge density and wavefunction files for a specific crystal. The charge density file is then used for subsequent calculations of electronic structure properties such as band structure and density of states. It is crucial to note that self-consistent field (SCF) calculations must precede electronic structure calculations such as band structure and density of states calculations. The charge density obtained from the SCF calculation is required for subsequent band structure and density of states calculations.

### 2.2.1 Input File Preparation for Self-Consistent Calculation of $Si$ Atom

The input files include the parameter file *scf.in* and the structure file *structure.as*. *scf.in* is shown below:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
#outputs
io.charge = true
io.wave = true
```

*scf.in* Input Parameters for Self-Consistent Field Calculation:

- `task`: Sets the calculation type; this calculation is a **scf** self-consistent field calculation.
- `cal.cutoffFactor`: Sets the coefficient for `cal.cutoff`. The cutoff energy used in the calculation is equal to `cal.cutoff` * `cal.cutoffFactor`.
- `io.charge`: Controls the output of the charge density file.
- `io.wave` : Controls the switch for outputting the wavefunction file;

The *structure.as* file is referenced as follows:

```
Total number of atoms
2
Lattice
0.00 2.75 2.75
2.75 0.00 2.75
```

(continues on next page)

(continued from previous page)

```
2.75 2.75 0.00
Direct
Si -0.125000000 -0.125000000 -0.125000000
Si 0.125000000 0.125000000 0.125000000
```

A standard self-consistent calculation usually takes the relaxed structure obtained from structural relaxation as the structural input.

**Note**

1. **To save ELF and potential data in structure relaxation and self-consistent calculations, simply set io.elf and io.potential to true;**
2. **To add a background charge to the system during the calculation, you can directly set the ``sys.electron`` parameter, which specifies the total number of valence electrons.**

### 2.2.2 Run the program.

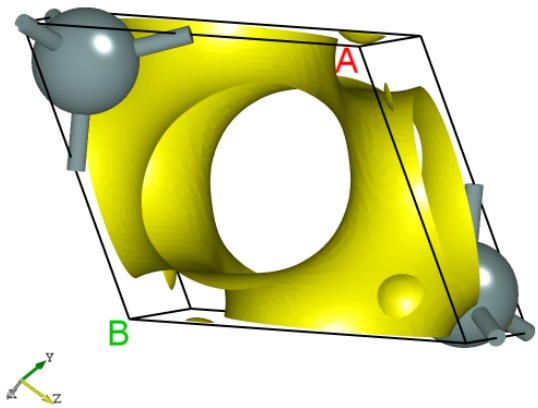
Once you have prepared the input files *scf.in* and *structure.as*, upload them to the server and run the *DS-PAW scf.in* calculation as described in Structure Relaxation.

### 2.2.3 Analysis of the calculation results

Based on the above input files, the following output files will be generated after the calculation is completed: *DS-PAW.log*, *scf.h5*, *rho.bin*, *wave.bin*, and *rho.h5*.

- *DS-PAW.log* : The log file obtained after the DS-PAW calculation, recording the main information such as energy iteration in the self-consistent calculation;
- *scf.h5* : The h5 output file for the self-consistent field (SCF) calculation. See *Output File Format Specification* for a structure analysis.
- *rho.bin* : Binary file of charge density, used for subsequent post-processing calculations;
- *rho.h5* : The **h5** format file of charge density, which can be easily converted to a format readable by VESTA (see *Auxiliary Tool User Guide*) for visualizing the charge density information.
- *wave.bin* : Binary file of wavefunctions, used for subsequent calculations;

The *rho.h5* file can be converted to a format supported by *VESTA* software via a Python script. See the *Auxiliary Tool User Guide* section for details. The processing yields 1D, 2D, and 3D charge density plots, with the 3D plot expected to look similar to the following:




## 2.3 band structure calculation

There are two common ways to perform band calculations: a two-step approach using *task=band* and a one-step approach using *task=scf*. This section will use the Si system as an example to illustrate the parameter settings for both methods.

### 2.3.1 $Si$ band structure calculation input file

#### 2.3.1.1 task = band two-step calculation

The input file contains the parameter files *scf.in* and *band.in*, the structure file *structure.as*. The *scf.in* settings are consistent with the self-consistent calculation in the previous section, and the *band.in* parameters are as follows:

```
# task type
task = band
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.iniCharge = ./rho.bin
cal.methods = 2
cal.smearing = 1
cal.cutoffFactor = 1.5
cal.totalBands = 12

#band related
band.kpointsLabel= [G,X,W,K,G,L]
band.kpointsCoord= [0, 0, 0, 0.5, 0, 0.5, 0.5, 0.25, 0.75, 0.375, 0.375, 0.75, 0, 0, 0, 0.5, 0.5, 0.5]
band.kpointsNumber= [30, 30, 30, 30, 30]
```

Introduction of input parameters for *band.in*:

In band calculations, you can generally retain the parameters from **sys.** and **cal.** in *band.in* and then set the specific parameters for the band structure calculation:

- `task`: Specifies the calculation type, which is a band structure calculation in this case.
- `cal.iniCharge`: Sets the path to the charge density file, supporting both absolute and relative paths. Here, ./ refers to the *rho.bin* file in the current directory.

A new set of band-related parameters has been added for band calculations, and these parameters are only effective during band calculations:

- `band.kpointsLabel`: Sets the labels for high-symmetry points during band structure calculation, one `band.kpointsLabel` corresponds to one `band.kpointsCoord`;
- `band.kpointsCoord`: Set the fractional coordinates of high-symmetry points for band structure calculations, with each group consisting of three numbers;
- `band.kpointsNumber`: Sets the number of k-points between every two adjacent high-symmetry points. There are two ways to set this parameter:
  - When the parameter is set as band.kpointsNumber= [30, 30, 30, 30, 30], the number of k-points between all high symmetry points is 30;
  - When band.kpointsNumber= [30] is set, the number of k-points between high symmetry points G and X is 30, and the k-point density is determined accordingly; uniform k-point sampling is then performed between high symmetry points X and W, W and K, K and G, and G and L. The actual number of k-points can be found in the parameter printing section of DS-PAW.log.
- `band.EfShift`: Determines whether to read the EFermi from rho.bin as the EFermi in the band calculation output. The default is true, which means reading EFermi from rho.bin.

*structure.as* file is the same as in the self-consistent calculation. (See Section 2.2)

**Note**

1. **When performing two-step calculations, the parameters cal.cutoffFactor and cal.cutoff in scf.in and band.in must be consistent, otherwise, a mismatch of grid data will occur.**
2. **cal.iniCharge specifies the path to the charge density file rho.bin generated by the SCF calculation.**

#### 2.3.1.2 task = scf: one-step calculation

The input file contains the parameter file *scf.in* and the structure file *structure.as*. The parameters for *scf.in* are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.totalBands = 12
cal.smearing = 1
cal.ksamping = G
```

(continues on next page)

(continued from previous page)

```
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
#outputs
io.charge = true
io.wave = true
#band related
io.band=true
band.kpointsLabel= [G,X,W,K,G,L]
band.kpointsCoord= [0, 0, 0, 0.5, 0, 0.5, 0.5, 0.25, 0.75, 0.375, 0.375, 0.75, 0, 0, 0,␣
↪0.5, 0.5, 0.5]
band.kpointsNumber= [30, 30, 30, 30, 30]
```

**Note**

1. For one-step band calculations, the result file is scf.h5. The band data is stored in the scf.h5 file, which can be directly processed by the bandplot.py script in *Auxiliary Tool User Guide*.
2. **io.band=true is only effective when task=scf.**
3. **When io.band is enabled, setting cal.iniCharge = ./rho.bin is no longer needed, and the calculation of high-symmetry points in k-space will be performed simultaneously during the scf calculation.**
4. **Two types of k-points need to be specified in the scf.in file: cal.kpoints for self-consistent field (SCF) calculations and band.kpoints parameters for band structure calculations. Both sets of k-points are required.**

### 2.3.2 Run the program.

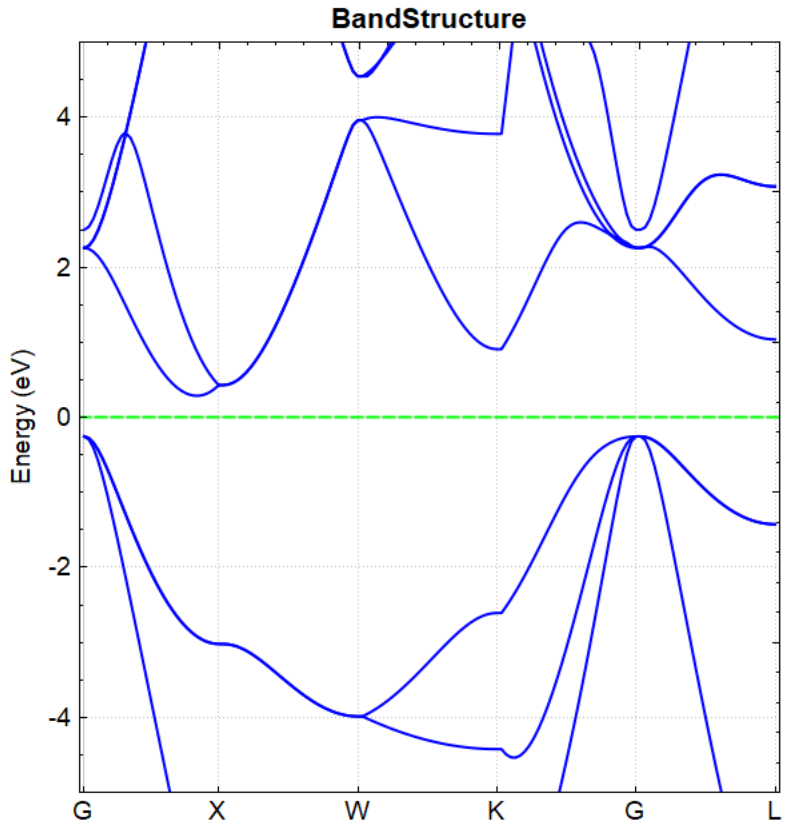
For the two-step calculation example, upload the parameter control files *scf.in*, *band.in*, and the structure file *structure.as* to the server. Then, execute *DS-PAW scf.in* and *DS-PAW band.in* sequentially, as described in the structure relaxation section.

### 2.3.3 Analysis of the calculation results.

Based on the input files mentioned above, the calculation will generate output files such as *DS-PAW.log*, *scf.h5*, and *band.h5*.

- *DS-PAW.log*: The log file obtained after the DS-PAW band structure calculation, which can be directly read to get important information such as band gap, VBM, and CBM.
- *band.h5* : The **h5** output file corresponding to the band calculation; it stores important data such as energy eigenvalues. The specific data structure is detailed in *Output File Format Specification*.

You can use **python** to process the data in *band.h5*. For detailed operations, see *Auxiliary Tool User Guide*. The resulting band structure plot should look like this:




**Note**

1. **The band diagrams obtained by the one-step and two-step band calculations are consistent.**

## 2.4 pband Projection of Band Calculation

Projected band structures refer to the decomposition of the energy at each k-point of each band into contributions from each atom and its orbitals during a band structure calculation.

### 2.4.1 *Si* Projected Band Structure Input File

The input files for the projected band structure calculation include the parameter file *pw_band.in*, the structure file *structure.as*, and the binary charge density file *rho.bin* obtained from the self-consistent calculation. *pw_band.in* is shown below:

```
# task type
task = band
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

```

(continues on next page)

(continued from previous page)

```
cal.iniCharge = ./rho.bin
cal.methods = 2
cal.smearing = 1
cal.cutoffFactor = 1.5
cal.totalBands = 12

#band related
band.kpointsLabel= [G,X,W,K,G,L]
band.kpointsCoord= [0, 0, 0, 0.5, 0, 0.5, 0.5, 0.25, 0.75, 0.375, 0.375, 0.75, 0, 0, 0, 0.5, 0.5, 0.5]
band.kpointsNumber= [30, 30, 30, 30, 30]
band.project=true
```

*pw_band.in* Input Parameters:

The projected band calculation differs from a regular band calculation in that the `band.project` parameter is set in the calculation parameters:

`band.project`: Controls whether projection calculations are performed in the band structure calculation;

### 2.4.2 Run the program

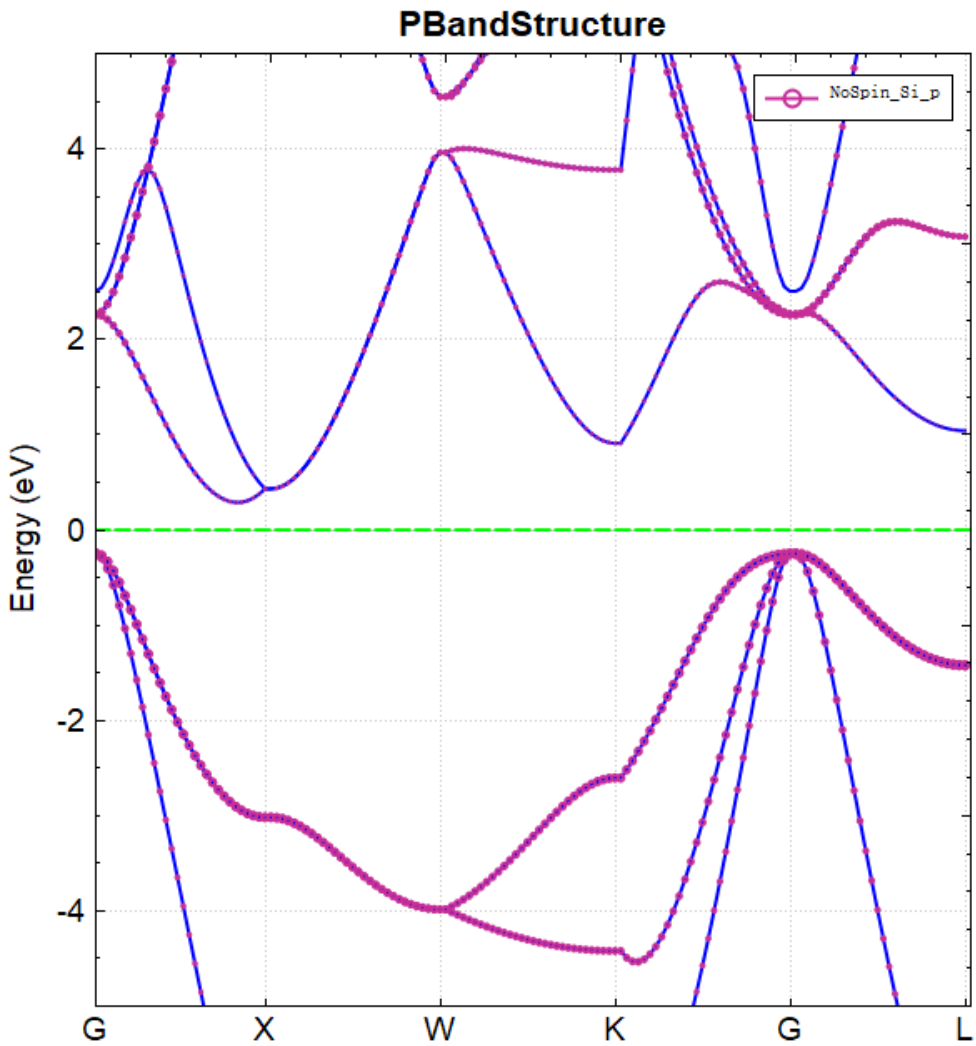
After preparing the input files *pw_band.in*, *structure.as*, and *rho.bin*, upload them to the server for execution. Run *DS-PAW pw_band.in* following the procedure described in the structural relaxation section.

### 2.4.3 Analysis Results

Based on the input files mentioned above, after the calculation is completed, output files such as *DS-PAW.log* and *band.h5* will be generated.

- *DS-PAW.log*: The log file generated after the DS-PAW band calculation;
- *band.h5* : The **h5** output file corresponding to the band structure calculation. Projected band data will also be saved in *band.h5*. See *Output File Format Specification* for details on the data structure;

Data processing of *band.h5* can be done using **python**, see *Auxiliary Tool User Guide* for details. The resulting band structure plot should look like this:




## 2.5 DOS calculation

Density of states (DOS) calculations can be performed in two ways: a two-step method with *task=dos* and a one-step method with *task=scf*. This section uses Si as an example to illustrate the parameter settings for both methods.

### 2.5.1 Input file for Density of States (DOS) calculation of a Si system

#### 2.5.1.1 task = dos two-step calculation

The input files include the parameter files *scf.in* and *dos.in*, and the structure file *structure.as*. *scf.in* is set consistently with the self-consistent calculation, and the parameters in *dos.in* are as follows:

```
# task type
task = dos
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
```

(continues on next page)

(continued from previous page)

```

cal.iniCharge = ./rho.bin
cal.methods = 2
cal.smearing = 4
cal.ksamping = G
cal.kpoints = [20, 20, 20]
cal.cutoffFactor = 1.5

#dos related
dos.range=[-10, 10]
dos.resolution=0.05
```

*dos.in* Input Parameters Introduction:

In the DOS calculation, parameters in **sys.** and **cal.** can be retained as much as possible in *dos.in*, then set the specific parameters for DOS calculation:

- `task`: Sets the calculation type. For this calculation, its DOS (density of states) calculation.
- `cal.iniCharge`: Sets the reading path for the charge density, supporting both absolute and relative paths; here, ./ refers to the *rho.bin* file in the current directory;
- `cal.kpoints`: Sets the k-point grid density. For DOS calculations, it is recommended to increase the k-points to about twice the density used in the self-consistent calculation.

A new set of parameters related to the density of states has been added for DOS calculations, and these parameters are only effective in the DOS calculation:

- `dos.range`: Sets the energy range for the density of states calculation.
- `dos.resolution`: Sets the energy interval precision for the density of states calculation. The number of points for the DOS calculation is the difference between **dos.range** divided by **dos.resolution** plus 1.

*structure.as* file is the same as the self-consistent calculation. (See Section 2.2)

**Note**

1. **When performing a two-step calculation, the parameters `cal.cutoffFactor` and `cal.cutoff` in `scf.in` and `dos.in` must be consistent; otherwise, grid data mismatch issues will occur.**

#### 2.5.1.2 task = scf one-step calculation

The input file includes the parameter file *scf.in*, the structure file *structure.as*, and the parameters for *scf.in* are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 4
```

(continues on next page)

(continued from previous page)

```
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
#outputs
io.charge = true
io.wave = true
#dos related
io.dos=true
dos.range=[-10, 10]
dos.resolution=0.05
```

**Note**

1. For the one-step DOS calculation, the result file is scf.h5. The DOS data is stored in the scf.h5 file, and you can directly use the *Auxiliary Tool User Guide*s dosplot.py script to process the scf.h5 file.
2. **io.dos=true is only effective when task=scf.**
3. **When io.dos is enabled, its no longer necessary to set cal.iniCharge = ./rho.bin; the DOS is obtained through the self-consistent calculation in this case.**

### 2.5.2 run the program

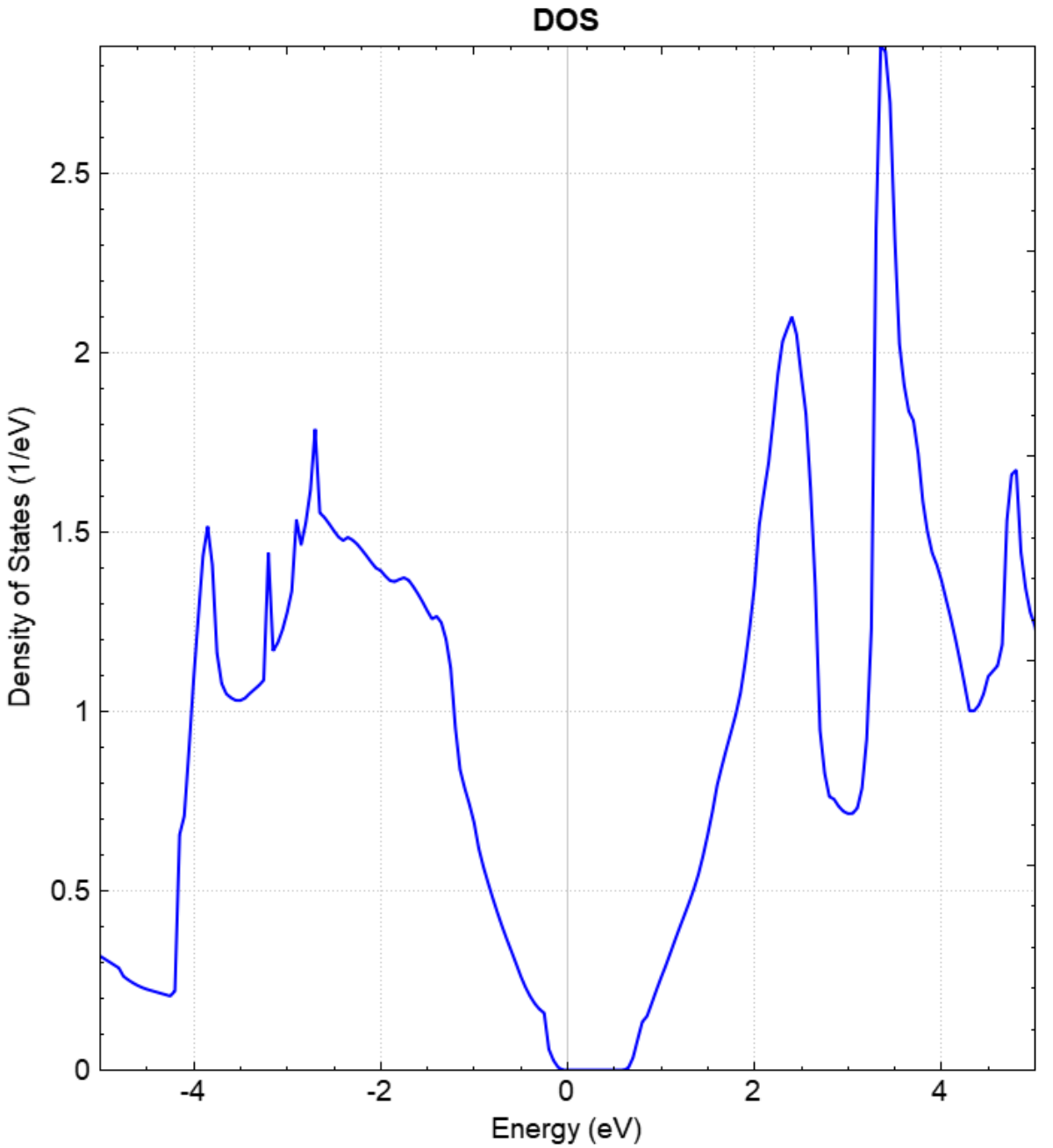
For the two-step calculation as an example, upload the parameter control files *scf.in*, *dos.in*, and the structure file *structure.as* to the server, and then sequentially run *DS-PAW scf.in* and *DS-PAW dos.in* as described in the structure relaxation section.

### 2.5.3 Analysis of the calculation results

Based on the input files mentioned above, the calculation will generate output files such as *DS-PAW.log*, *scf.h5*, and *dos.h5*.

- *DS-PAW.log* : Log file generated after DS-PAW density of states calculation;
- *dos.h5* : The h5 file containing the density of states data. For details on its structure, see the *Output File Format Specification* section.

You can process *dos.h5* data using **python**. See the *Auxiliary Tool User Guide* section for details. The resulting density of states plot should look like this:




## 2.6 pdos Projected Density of States Calculation

The calculation of projected density of states refers to the process of expanding the density of states at each energy level during the density of states calculation into contributions from each atom and its orbitals.

### 2.6.1 $Si$ projected density of states calculation input file

The input files for projected density of states calculations include the parameter file *pdos.in*, the structure file *structure.as*, and the charge density file from the self-consistent calculation *rho.bin*. The *pdos.in* file is as follows:

```
# task type
task = dos
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.iniCharge = ./rho.bin
cal.methods = 2
cal.smearing = 4
cal.ksamping = G
cal.kpoints = [20, 20, 20]
cal.cutoffFactor = 1.5

#dos related
dos.range=[-10, 10]
dos.resolution=0.05
dos.project = true
```

Introduction to the input parameters for *pdos.in*:

The difference between projected density of states and regular density of states lies in the setting of the `dos.project` parameter within the calculation parameters:

- `dos.project` : Controls the switch for projected calculations in the density of states calculation.

### 2.6.2 Run the program

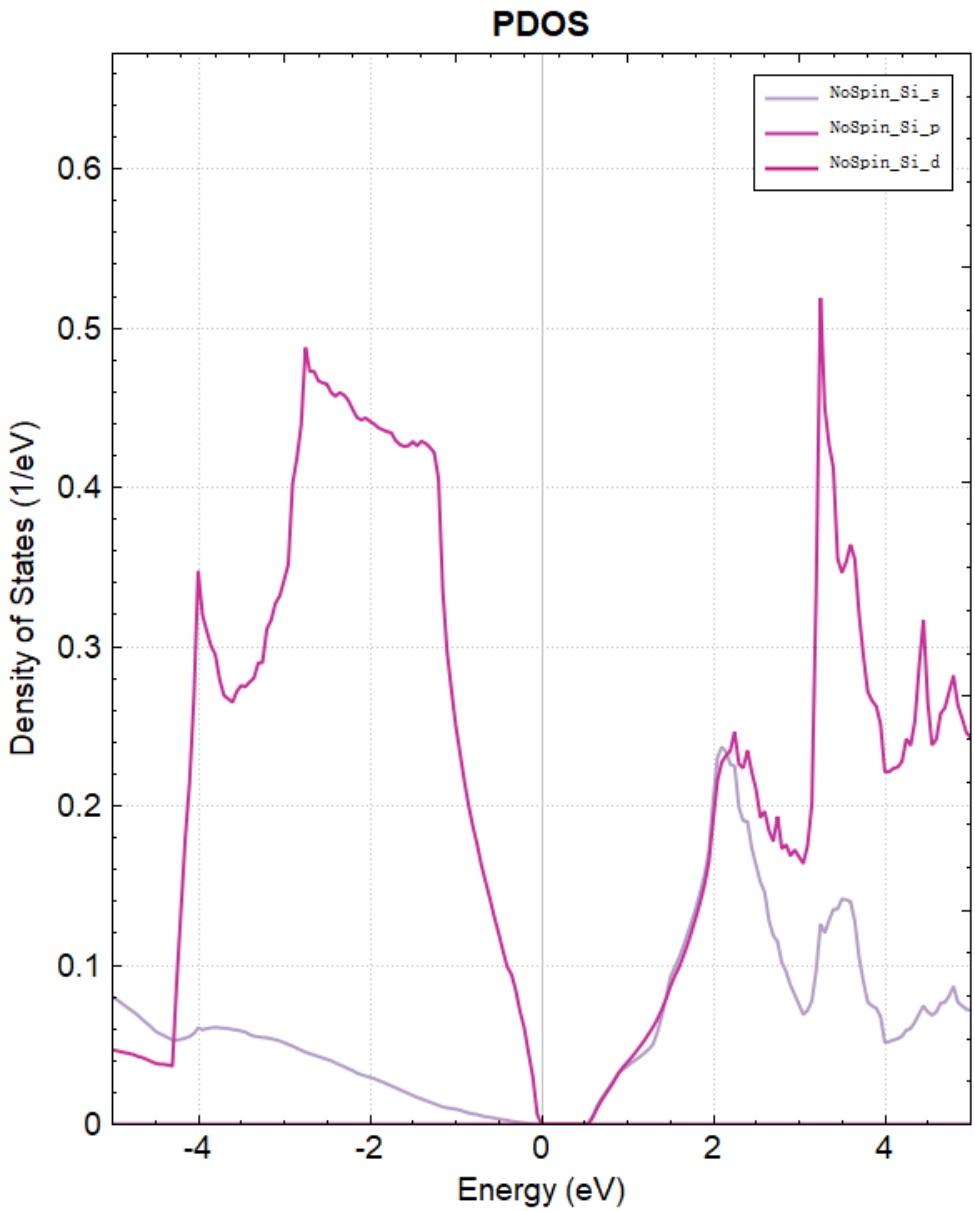
After preparing the input files *pdos.in*, *structure.as*, and *rho.bin*, upload the files to the server and run *DS-PAW pdos.in* as described in the structure relaxation method.

### 2.6.3 Analysis of the calculation results.

Based on the above input files, the calculation will generate output files such as *DS-PAW.log* and *dos.h5*.

- *DS-PAW.log* : The log file generated after the DS-PAW density of states calculation;
- *dos.h5* : The **h5** output file corresponding to the density of states calculation; the projected density of states data is stored in the *dos.h5* file. For the specific data structure, see the *Output File Format Specification* section;

You can process *dos.h5* data using **python**. See *Auxiliary Tool User Guide* for specific operations. The resulting projected density of states plot should look like the following:




## 2.7 potential calculation

There are two methods for calculating the potential function: a two-step method using *task=potential* and a one-step method using *task=scf*. This section takes the Si system as an example to introduce the corresponding parameter settings for both methods.

### 2.7.1 Input file for $Si$ potential function calculation

#### 2.7.1.1 task = potential two-step calculation

The input files include the parameter file *scf.in*, *potential.in*, and the structure file *structure.as*. *scf.in* is set up consistently with the self-consistent calculation, while *potential.in* is configured as follows:

```
# task type
task = potential
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.iniCharge = ./rho.bin
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5

#potential related
potential.type=all
```

Introduction of the input parameters in *potential.in*:

In the potential calculation, you can retain as many parameters as possible from **sys.** and **cal.** in *potential.in*, and then set the specific parameters for the potential calculation:

- `task`: Set the calculation type. This calculation is a `potential` potential function calculation.
- `cal.iniCharge`: Sets the path to read the charge density file, supporting both absolute and relative paths. Here, ./ refers to the *rho.bin* file in the current directory;

New parameters in potential calculation:

- `potential.type`: Controls the type of potential saved. When **all** is selected, both the electrostatic potential (sum of ionic and Hartree potentials) and the local potential (sum of electrostatic and exchange-correlation potentials) will be saved after the potential calculation is completed.

*structure.as* file as the self-consistent calculation result. (See Section 2.2)

**Note**

1. **When performing two-step calculations, the parameters `cal.cutoffFactor` and `cal.cutoff` in both `scf.in` and `potential.in` must be consistent; otherwise, a mismatch in the grid data will occur.**
2. **If the system being calculated requires dipole correction, the user needs to add the parameters `corr.dipol = true` and `corr.dipolDirection` in both the self-consistent field (SCF) and potential calculation input files. `corr.dipol = true` enables the dipole correction switch, and `corr.dipolDirection` sets the dipole correction direction; a, b, and c represent the directions along the lattice vectors a, b, and c, respectively.**
3. **For a specific example of dipole correction, see the application case: Calculation of Work Function for Au-Al System.**

#### 2.7.1.2 task = scf one-step calculation

The input file contains the parameter file *scf.in*, the structure file *structure.as*, and the parameters for *scf.in* are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
#outputs
io.charge = true
io.wave = true
#potential related
io.potential = true
```

**Note**

1. For the one-step potential calculation, the corresponding result file is scf.h5. In this case, the potential data is stored in the scf.h5 file, and you can directly call the potential processing script of *Auxiliary Tool User Guide* to analyze the scf.h5 file.
2. **io.potential=true is only effective when task=scf.**

### 2.7.2 Run the program.

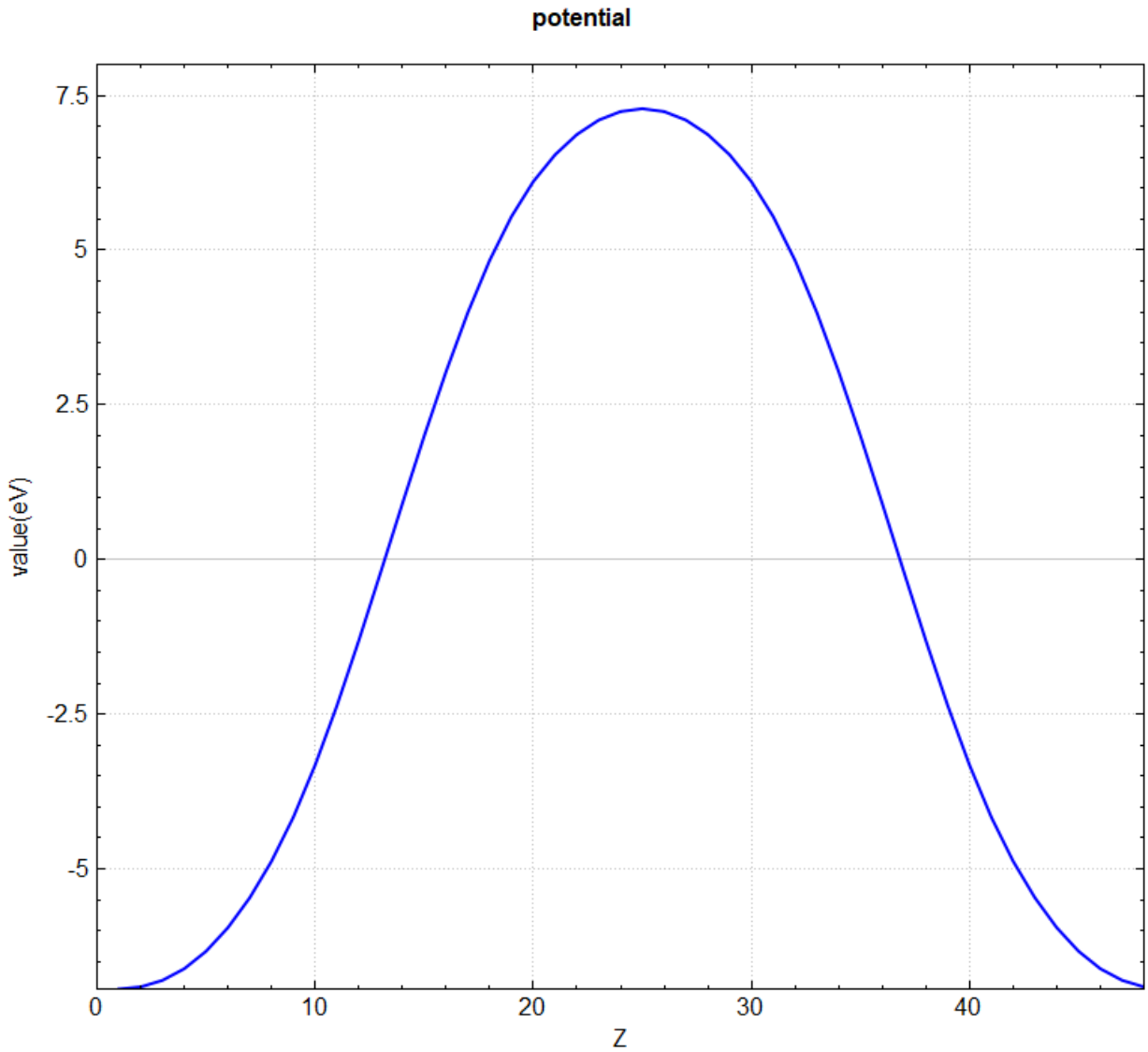
For the two-step calculation as an example, upload the parameter control file *scf.in*, *potential.in*, and structure file *structure.as* to the server, and then execute *DS-PAW scf.in* and *DS-PAW potential.in* sequentially according to the method described in structure relaxation.

### 2.7.3 Analysis of the calculation results

Based on the input files mentioned above, the calculation will generate the following output files: *DS-PAW.log*, *scf.h5*, and *potential.h5*.

- *DS-PAW.log* : Log file generated after DS-PAW potential calculation.
- *potential.h5* : The **h5** output file corresponding to the potential calculation, with specific structure detailed in *Output File Format Specification*.

You can use a **Python** script to convert the *potential.h5* format to a format supported by **VESTA** software, or directly use the script to perform in-plane averaging of the 3D potential function. For specific operations, see the *Auxiliary Tool User Guide* section. The processed vacuum direction potential curve is shown below:




## 2.8 elf calculation of electronic local density

There are two ways to calculate the electron localization function (ELF): a two-step approach with *task=elf* and a one-step approach with *task=scf*. This section uses a Si system as an example to illustrate the corresponding parameter settings for both methods.

### 2.8.1 $Si$ Electronic Localized Function calculation input file

#### 2.8.1.1 task = elf two-step calculation

Input files include parameter files *scf.in* and *ELF.in*, and structure file *structure.as*. *scf.in* settings are consistent with self-consistent calculations, while *ELF.in* settings are as follows:

```
# task type
task = elf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.iniCharge = ./rho.bin
cal.methods = 2

cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
```

*ELF.in* Input Parameter Description:

In ELF calculations, it is recommended to retain the **sys.** and **cal.** parameters in *ELF.in* as much as possible:

- `task`: Sets the calculation type; this calculation is an ELF calculation.
- `cal.iniCharge`: Sets the reading path of the charge density file. Both absolute and relative paths are supported. Here, ./ represents the *rho.bin* file in the current path;

The *structure.as* file is the same as that used in the self-consistent calculation (see Section 2.2).

**Note**

1. **For two-step calculations, the parameters `cal.cutoffFactor` and `cal.cutoff` in both `scf.in` and `ELF.in` must be consistent; otherwise, grid data mismatch will occur.**
2. **ELF calculation does not support non-collinear calculations.**

#### 2.8.1.2 task = scf one-step calculation

The input files include the parameter file *scf.in*, the structure file *structure.as*. The *scf.in* parameters are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
#outputs
io.charge = true
io.wave = true
```

(continues on next page)

(continued from previous page)

```
#elf related
io.elf = true
```

**Note**

1. The output file for a single-step calculation of electron localization density is scf.h5. The electron localization density data is stored in this file and can be directly analyzed using the electron localization density processing scripts in *Auxiliary Tool User Guide*.
2. *io.elf=true* only takes effect when *task=scf*.
3. ELF calculation does not support non-collinear calculations.

### 2.8.2 Run the program

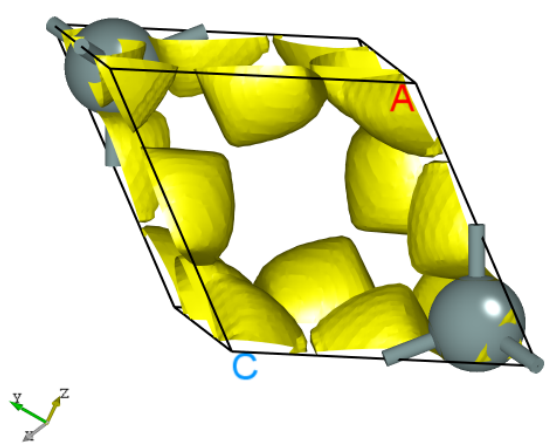
For a two-step calculation example, upload the parameter control files *scf.in*, *ELF.in*, and the structure file *structure.as* to the server. Then, execute *DS-PAW scf.in* and *DS-PAW ELF.in* sequentially, as described in the structural relaxation section.

### 2.8.3 Analysis Results

Based on the input files above, after the calculation is completed, output files such as *DS-PAW.log*, *scf.h5*, and *elf.h5* will be generated.

- *DS-PAW.log* : Log file generated after the DS-PAW local density calculation.
- *elf.h5* : ELF calculation output file in **h5** format; details of the structure are described in *Output File Format Specification*.

The *elf.h5* format can be converted to a format supported by the **VESTA** software using a **python** script; see the *Auxiliary Tool User Guide* section for details. The resulting 3D electron localization density map should look like this:



## 2.9 pcharge Part Charge Density Calculation

This section will analyze the charge density of specific bands at specified k-points, using graphene as an example. It details the preparation of partial charge density calculations after self-consistent field calculations, and the subsequent analysis through plotting.

### 2.9.1 Input file for partial charge density calculation of graphene

The input files include the parameter file *pcharge.in* and the structure file *structure.as*, the charge density file *rho.bin* and the wavefunction file *wave.bin* obtained from self-consistent calculations, and *pcharge.in* is as follows:

```
# task type
task = pcharge
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [9, 9, 1]
cal.cutoffFactor = 1.5
cal.iniCharge = ./rho.bin
cal.iniWave = ./wave.bin

#pcharge related
pcharge.bandIndex = [4,5]
pcharge.kpointsIndex = [12]
pcharge.sumK= false
```

*pcharge.in* input parameters introduction:

In the partial charge density calculation, parameters from *sys.* and *cal.* can be retained in *pcharge.in* as much as possible, and then the specific parameters for partial charge density calculation can be set:

- `task` : Sets the calculation type, which is partial charge density calculation in this case;
- `cal.iniCharge` : Sets the path for reading the charge density file, supporting absolute and relative paths. Here, ./ refers to the *rho.bin* file in the current directory.
- `cal.iniWave` : Sets the reading path for the wave function file, supporting both absolute and relative paths. Here, ./ indicates the *wave.bin* file under the current path;
- `pcharge.bandIndex` : Specifies the band indices for charge density analysis. Here, [4,5] indicates that the charge density of band 4 and band 5 will be analyzed.
- `pcharge.kpointsIndex` : Sets the K-point index for charge density analysis. Here, [12] indicates that the K-point index is 12 when analyzing the charge density of two bands.
- `pcharge.sumK` : Controls whether to sum the band data of all analyzed K-points. Here, false means no summation.

*structure.as* file is referenced as follows:

```
Total number of atoms
2
```

(continues on next page)

(continued from previous page)

```
Lattice
2.46120000 0.00000000 0.00000000
-1.23060000 2.13146172 0.00000000
0.00000000 0.00000000 6.70900000
Cartesian
C  0.61530000  0.35524362  3.35450000
C  0.61530000  1.77621810  3.35450000
```

**Note**

1. **Partial charge density calculation is performed in two steps, with the second step requiring reading the charge density file rho.bin and the wavefunction file wave.bin from the self-consistent calculation.**

### 2.9.2 run program execution

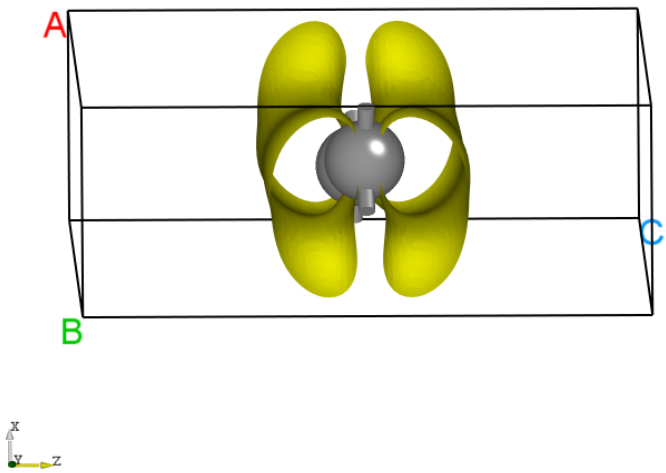
Prepare the input files *pcharge.in*, *structure.as*, and the self-consistent calculation results *rho.bin* and *wave.bin*, upload them to the server for execution, and run *DS-PAW pcharge.in* following the method described in structure relaxation.

### 2.9.3 Analysis of the calculation results

Based on the input files described above, the calculation will generate output files such as *DS-PAW.log* and *pcharge.h5*.

- *DS-PAW.log* : The log file generated after the DS-PAW partial charge density calculation.
- *pcharge.h5* : The HDF5 data file after the partial charge density calculation is completed. The charge density data for two bands is saved in *pcharge.h5* at this time. The specific data structure can be found in the *Output File Format Specification* section.

You can process the data in *pcharge.h5* using **python**. See *Auxiliary Tool User Guide* for specific operations. The charge density plot for band **4** at k-point index **12** should look like this:




## 2.10 hse hybrid functional calculation

This section will demonstrate the calculation of hybrid functional band structures using the direct band structure calculation method within the DS-PAW code, using the Si system as an example. We will observe the changes in the band gap after performing hybrid functional calculations.

### 2.10.1 $Si$ Hybrid Functional Calculation Input File

The input file includes the parameter file *ioband.in* and the structure file *structure.as*. The content of *ioband.in* is as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.spin = none
#scf related
cal.methods = 1
cal.totalBands = 12
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [5, 5, 5]
cal.cutoffFactor = 1.5
#band related
```

(continues on next page)

(continued from previous page)

```
io.band = true
band.kpointsCoord=[0.62500000,0.25000000,0.62500000,0.50000000,0.00000000,0.50000000,0.
↪00000000,0.00000000,0.00000000,0.50000000,0.00000000,0.50000000,0.50000000,0.25000000,
↪0.75000000,0.37500000,0.37500000,0.75000000,0.00000000,0.00000000,0.00000000]
band.kpointsLabel = [U,X,G,X,W,K,G]
band.kpointsNumber = [20,20,20,20,20,20]
band.project = false
#HSE related
sys.hybrid=true
sys.hybridType=HSE06
#outputs
io.charge = true
io.wave = true
```

*ioband.in* Input Parameters:

In hybrid functional calculations, you can generally preserve the *sys.* and *cal.* parameters in *ioband.in* as much as possible, and then set the specific parameters for hybrid functional calculations:

- `sys.hybrid` : Controls the switch for hybrid functional calculations. *true* indicates the introduction of hybrid functional calculations;
- `sys.hybridType` : Sets the type of hybrid functional, which is HSE06 in this case;

*structure.as* file is the same as in the self-consistent calculation. (See Section 2.2)

**Note**

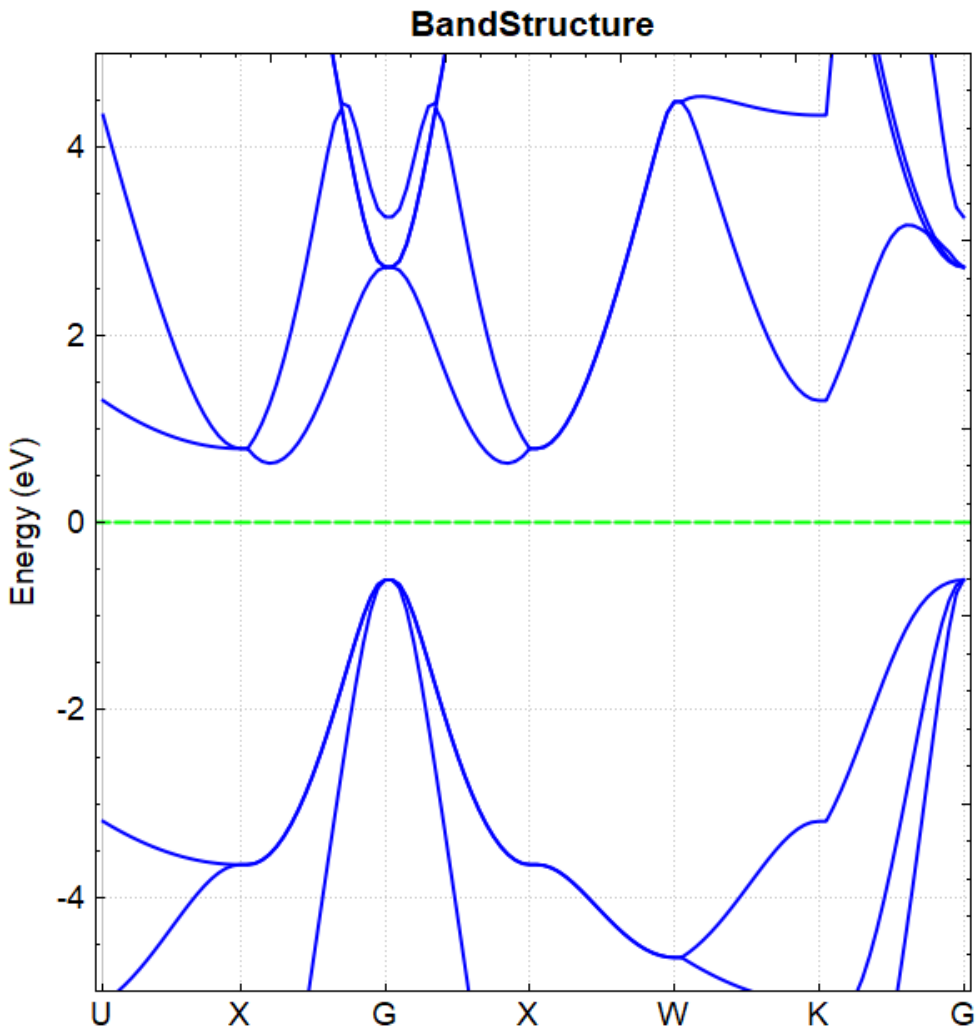
1. **Unlike regular calculations where the functional type is set using `sys.functional`, hybrid functional calculations control the hybrid functional type via the `sys.hybridType` parameter.**
2. **Hybrid functional calculations only support tasks scf and relax. Therefore, band structure calculations with hybrid functionals can only be performed in a single-shot manner.**
3. **It is recommended to use damped MD/conjugated gradient methods for electronic self-consistent field (SCF) calculations with hybrid functionals, corresponding to setting the parameter cal.methods = 4/5.**
4. **Hybrid functional calculations can also use the block Davidson method for electronic self-consistent calculations, i.e., cal.methods = 1 in this example. In this case, the scf.mixType parameter will default to Kerker.**

### 2.10.2 Run the program.

Prepare the input files *ioband.in* and *structure.as* and upload them to the server to run. Execute *DS-PAW ioband.in* as described in Structure Relaxation.

### 2.10.3 Analysis of calculation results

After the calculation is completed based on the input files mentioned above, output files such as *DS-PAW.log* and *scf.h5* will be generated. The method for processing *scf.h5* is the same as the band structure calculation method (see Section 2.3), and the resulting band structure plot should look like the following:



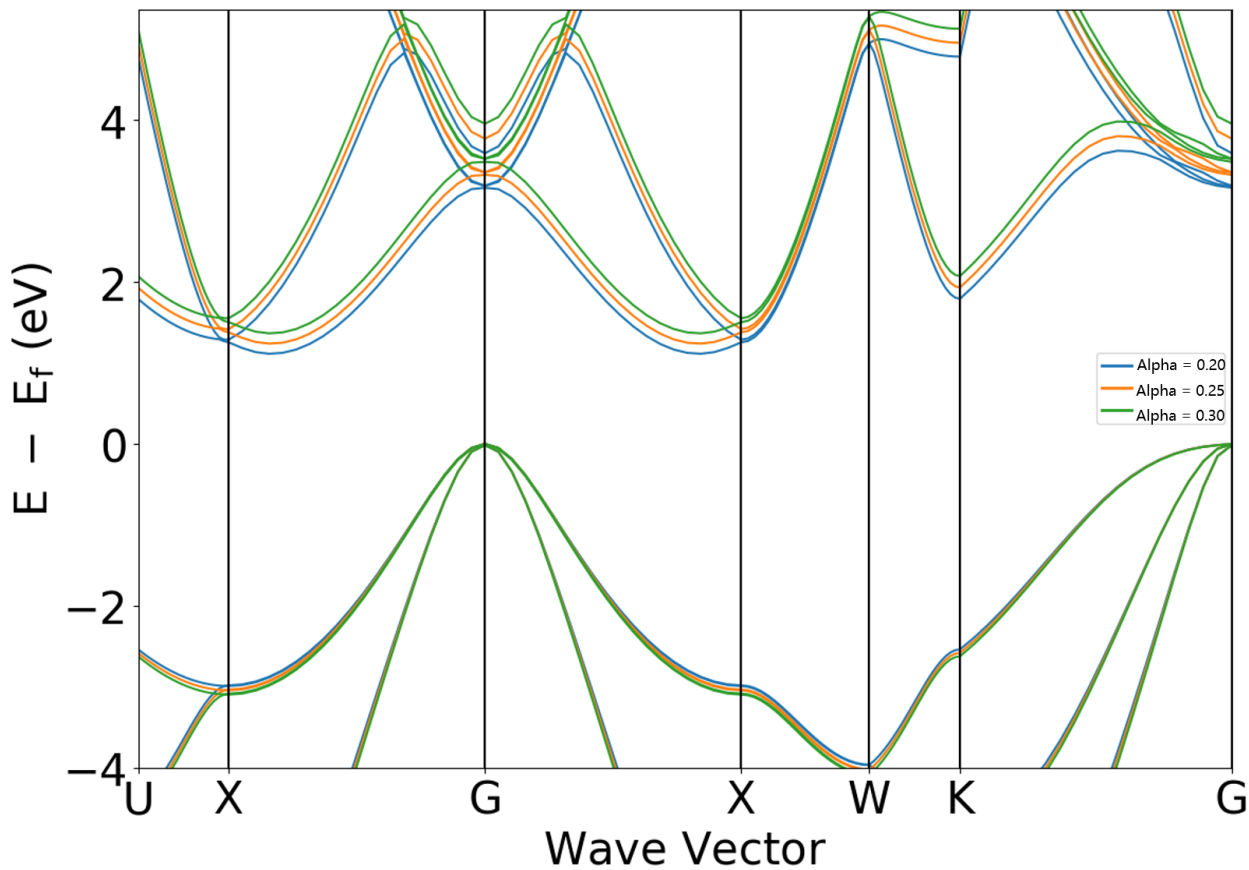

The figure shows that the band gap between the valence band and the conduction band increases after performing the hybrid functional calculation, approximately to **1.2394** eV, while the band gap obtained without the hybrid functional calculation is approximately **0.6433** eV.

### 2.10.4 Modifying the hybrid functional Alpha coefficient

The hybrid functional method shown in Section 2.10.1 is HSE06, with a corresponding hybrid functional coefficient of `sys.hybridAlpha = 0.25`. Adjust the `sys.hybridAlpha` parameter and perform the following two calculations:

- Modify the parameter in *scf.in* and *band.in*: `sys.hybridAlpha = 0.20`
- Modify the parameter in *scf.in* and *band.in*: `sys.hybridAlpha = 0.30`

Obtain the following band structure comparison:




Analysis of the figure shows that increasing the `sys.hybridAlpha` coefficient leads to a further increase in the band gap. The band gap values of **1.1146**, **1.2394**, and **1.3665** eV can be read from the *DS-PAW.log* file when `sys.hybridAlpha` is set to **0.20**, **0.25**, and **0.30**, respectively.

## 2.11 van der Waals Correction Calculation

This section will use the structural relaxation of a graphite system as an example to illustrate how to correctly set up van der Waals corrections in DS-PAW, and will compare and analyze the results with and without the van der Waals correction.

### 2.11.1 Graphite structure relaxation input file

When relaxing graphite, you can choose to correct van der Waals forces using a semi-empirical method or a functional correction method. The parameter settings for both methods are described below.

#### 2.11.1.1 Empirical Correction

The input files include the parameter file *relax.in* and the structure file *structure.as*. *relax.in* is shown below:

```
# task type
task = relax
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [21, 21, 7]
cal.cutoff  = 600
scf.convergence = 1.0e-05
#relax related
relax.max =  60
relax.freedom = all
relax.convergence =  0.01
relax.methods = CG
#vdw related
corr.VDW = true
corr.VDWType = D3G
```

*relax.in* Input Parameters:

In the van der Waals correction calculation, try to keep the parameters of sys. and cal. in *relax.in*, then set the parameters specific to the van der Waals correction calculation.

- `corr.VDW` : Controls the switch for the semi-empirical van der Waals correction, true indicates it is turned on;
- `corr.VDWType`: Sets the type of van der Waals correction, D3G representing the DFT-D3 method of Grimme;

The *structure.as* file is referenced as follows:

```
Total number of atoms
4
Lattice
2.46729136 0.00000000 0.00000000
-1.23364568 2.13673699 0.00000000
0.00000000 0.00000000 7.80307245
Cartesian
C 0.00000000 0.00000000 1.95076811
C 0.00000000 0.00000000 5.85230434
C 0.00000000 1.42449201 1.95076811
C 1.23364689 0.71224492 5.85230434
```

**Note**

1. **When correcting van der Waals forces using semi-empirical methods, different types of exchange-correlation functionals can be selected. The selectable values for sys.functional are PBE/REVPBE/RPBE/PBESOL.**

2. **DS-PAW supports using semi-empirical methods to correct van der Waals forces while simultaneously enabling hybrid functional calculations.**

#### 2.11.1.2 Functional Correction

The input file corresponding to the functional correction, *relax.in*, can be structured as follows:

```
# task type
task = relax
#system related
sys.structure = structure.as
sys.symmetry = true
sys.spin = none
#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [21, 21, 7]
cal.cutoff  = 600
scf.convergence = 1.0e-05
#relax related
relax.max =  60
relax.freedom = all
relax.convergence =  0.01
relax.methods = CG
#vdw related
sys.functional = vdw-optPBE
```

*relax.in* Input Parameter Introduction:

In calculations with van der Waals corrections, parameters related to *sys.* and *cal.* can generally be kept in the *relax.in* file. Subsequently, set the specific parameters for the van der Waals correction calculation.

- `sys.functional`: Controls the type of functional. When selecting a functional that includes van der Waals correction, simply set the vdw-series functional parameters. This example uses the vdw-optPBE functional. Supported functional types are listed in the *Parameters Explanation* section.

**Note**

1. **From a theoretical perspective, there are two different ways to correct for van der Waals interactions, corresponding to the parameters corr.VDW = true (semi-empirical correction) and sys.functional = vdw- (functional correction), respectively.**

### 2.11.2 run the program

For the example of a semi-empirical correction, after preparing the input file, upload the *relax.in* and *structure.as* files to the server and run the *DS-PAW relax.in* file as described in structure relaxation.

### 2.11.3 Analysis of calculation results

After the calculation based on the above input file, output files such as *DS-PAW.log*, *relax.h5*, and *latestStructure.as* will be generated. (Another set of calculations without considering van der Waals corrections is added for comparison.)

Drag *latestStructure.as* into Device Studio to view the structure. The lattice constants after relaxation are shown in the following table. By comparison, it is found that the value of the lattice vector **c** obtained from structural relaxation with the addition of van der Waals correction is closer to the experimental results reported in :footcite:p:*Rgo2015ComparativeSO*.

| Procedure | a (Å) | c (Å) |
|---|---|---|
| vdw-D3G this work | 2.463 | 6.954 |
| PBE this work | 2.464 | 7.914 |
| Experiment | 2.462 | 6.707 |

## 2.12 Optical Property Calculations

There are two ways to perform optical calculations: a two-step approach with *task=optical* and a one-step approach with *task=scf*. This section will use the Si system as an example to illustrate how to calculate optical properties in DS-PAW and analyze a series of optical properties by plotting them.

### 2.12.1 $Si$ optical property calculation input file

#### 2.12.1.1 task = optical two-step calculation

The input files contain the parameter file *scf.in*, *optical.in*, and the structure file *structure.as*. The settings in *scf.in* are consistent with the self-consistent calculation, and the settings in *optical.in* are as follows:

*optical.in* is set as follows:

```
# task type
task = optical
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [12, 12, 12]
cal.cutoffFactor = 1.5
cal.iniCharge = ./rho.bin

#optical related
optical.grid = 2000
optical.sigma = 0.05
optical.smearing = 1
```

In optical calculations, you can retain the parameters of *sys.* and *cal.* as much as possible in *:guilabel: `optical.in* and then set the parameters specific to optical calculations:

- `task` : Sets the calculation type, this calculation is `task = optical` : optical calculation;
- `cal.iniCharge` : Sets the path to read the charge density file, supporting both absolute and relative paths. Here, ./ indicates the *rho.bin* file in the current directory;
- `optical.grid` : Specifies the number of grid points within the energy range for DS-PAW optical property calculations, in this case, 2000.

- `optical.sigma`: Determines the broadening width when using the broadening algorithm specified by `optical.smearing`, which is 0.05 in this example.
- `optical.smearing`: Specifies the smearing algorithm used for energy broadening in optical calculations, which is 1 in this case.

#### 2.12.1.2 task = scf one-step calculation

The input file contains the parameter file *scf.in* and the structure file *structure.as*. The settings for *scf.in* are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [12, 12, 12]
cal.cutoffFactor = 1.5
#optical related
io.optical = true
```

*scf.in* Input parameters introduction:

In optical property calculations, you can retain as many *sys.* and *cal.* parameters as possible in *scf.in*, and then set the specific parameters for optical property calculations:

- `io.optical`: Controls the switch for optical property calculations. When io.optical=true, the system performs calculations for optical properties;

*structure.as* file as in self-consistent calculation. (See Section 2.2)

### 2.12.2 run the program

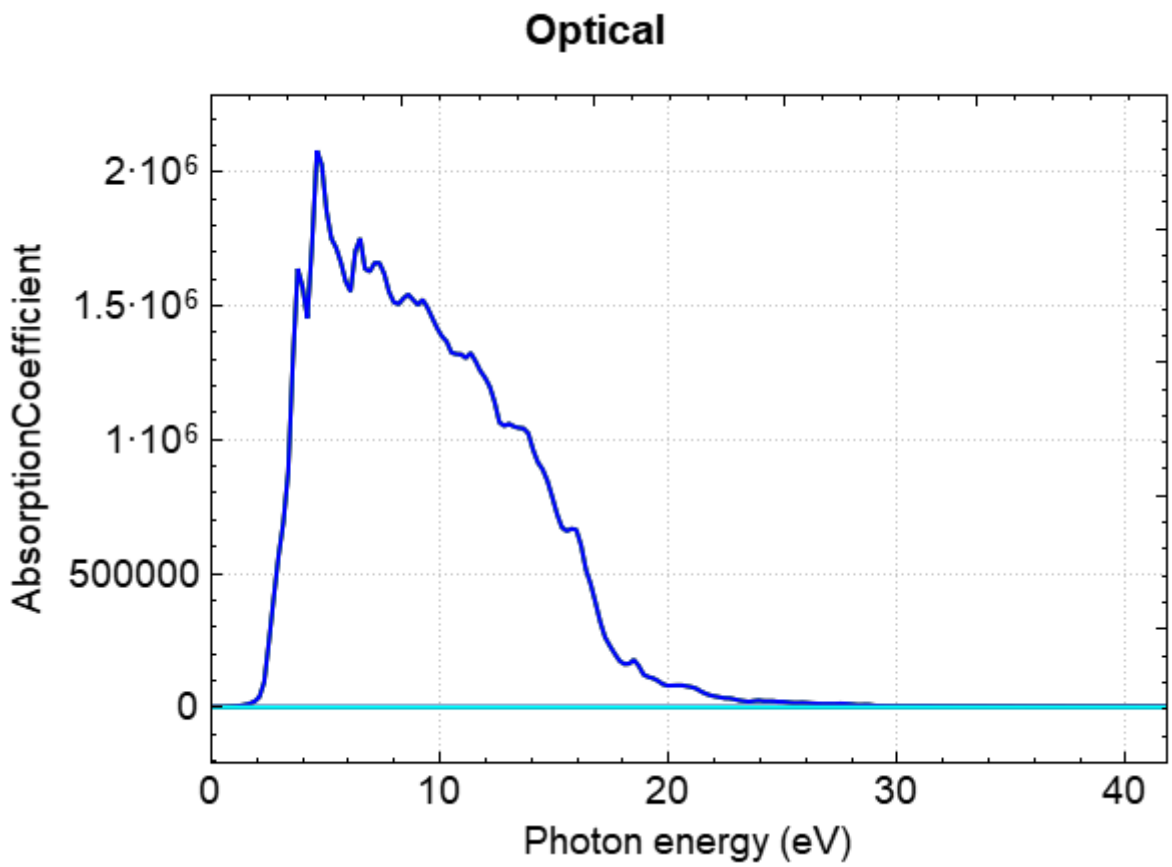
For the two-step calculation as an example, after preparing the input files, upload the *scf.in*, *optical.in*, and *structure.as* files to the server and run them. Execute *DS-PAW scf.in* and *optical.in* as described in the structure relaxation.

### 2.12.3 Analysis of calculation results

Based on the input files mentioned above, the calculation will generate output files including *DS-PAW.log*, *scf.h5*, and *optical.h5*.

- *DS-PAW.log* : Log file generated after DS-PAW optical properties calculation.
- *optical.h5* : The **h5** data file after the optical properties calculation is completed. Note that the name of the h5 file is strictly consistent with the task type. For the data structure of the h5 file, please refer to *Output File Format Specification*.

You can use **python** to process the data from *optical.h5* or the one-step calculation result *scf.h5*. For specific operations, refer to the *Auxiliary Tool User Guide* section. Processing allows you to obtain curves of the real part of the dielectric function, the imaginary part of the dielectric function, the absorption coefficient, the extinction coefficient, the conductivity, the reflectivity, the refractive index, and the energy loss as a function of energy. Taking the absorption coefficient curve as an example, the resulting curve should look like the following:




## 2.13 Frequency Calculation

This section will use the $CO$ molecule as an example to illustrate how to perform frequency calculations in DS-PAW.

### 2.13.1 $CO$ frequency calculation input file

The input file contains the parameter file *frequency.in* and the structure file *structure.as*, with *frequency.in* as follows:

```
# task type
task = frequency
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = MP
cal.kpoints = [9, 9, 9]
cal.cutoffFactor = 1.5
scf.convergence = 1.0e-6
#frequency related
```

(continues on next page)

(continued from previous page)

```
frequency.dispOrder = 1
frequency.dispRange = 0.02
#outputs
io.charge = false
io.wave = false
```

Introduction to input parameters for *frequency.in*:

In the frequency calculation, you can retain as many parameters from *sys.* and *cal.* as possible in *frequency.in*, and then set the parameters specific to the frequency calculation:

- `task` : Set the calculation type, which is frequency calculation for this run;
- `frequency.dispOrder` : Sets the atomic vibration mode for frequency calculations. 1 corresponds to the central difference method, i.e., 2 atomic vibration modes: the displacement of atoms in each Cartesian direction is ±`frequency.dispRange`; 2 corresponds to 4 atomic vibration modes: the displacement of atoms in each Cartesian direction is ±`frequency.dispRange` and ±`2*frequency.dispRange`;
- `frequency.dispRange` : Sets the displacement magnitude of atoms during frequency calculation.

The *structure.as* file is referenced as follows:

```
Total number of atoms
2
Lattice
8.0 0.0 0.0
0.0 8.0 0.0
0.0 0.0 8.0
Cartesian  Fix_x Fix_y Fix_z
O 0 0 0       T T F
C 0 0 1.143   T T F
```

**Note**

1. **Increase the convergence accuracy of the self-consistent field (SCF) calculation during frequency calculation. It is recommended to set it to 1.0e-6 or higher.**
2. **Since C and O atoms are fixed in the x and y directions, they can only move in the z direction.**

### 2.13.2 run program execution

After preparing the input files, upload the *:guilabel:`frequency.in* and *:guilabel:`structure.as* files to the server and run *:guilabel:`DS-PAW frequency.in* as described in Structure Relaxation.

### 2.13.3 Analysis of calculation results

Based on the input files mentioned above, the calculation will generate output files including *DS-PAW.log*, *frequency.h5*, and *frequency.txt*.

- *DS-PAW.log* : The log file generated after the DS-PAW frequency calculation.
- *frequency.h5* : The h5 data file after frequency calculation. The frequency data is stored in this file at this time. For the specific data structure, see the *Output File Format Specification* section.
- *frequency.txt*: The **txt** text file generated after frequency calculation, which writes frequency-related data. This file contains the same data as *frequency.h5*, making it easy for users to quickly access the information.

The following data can be obtained from *frequency.txt*:

| Frequency | THz | 2PiTHz | cm-1 | meV |
|---|---|---|---|---|
| 1 f | 63.844168 | 401.144726 | 2129.612084 | 264.038342 |
| 2 f/i | 0.051335 | 0.322546 | 1.712346 | 0.212304 |

CO moves only along the z-axis for two atoms, resulting in only two frequencies. Based on the table above, one vibrational mode has a frequency of approximately **63.8** THz, and the other is a near-zero imaginary frequency. Generally, imaginary frequencies less than 2 THz can be considered negligible.

## 2.14 Calculating elastic constants

This section will use the Si system as an example to demonstrate how to perform elastic calculations in DS-PAW.

### 2.14.1 $Si$ Input file for elastic constant calculation

The input file includes the parameter file *elastic.in* and the structure file *structure.as*, with *elastic.in* as follows:

```
# task type
task = elastic
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none
#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [5, 5, 5]
cal.cutoffFactor = 1.5
scf.convergence = 1.0e-6
#frequency related
elastic.dispOrder = 1
elastic.dispRange = 0.01
#outputs
io.charge = false
io.wave = false
```

*elastic.in* Input Parameter Introduction:

In elastic calculations, you can generally preserve the *sys.* and *cal.* parameters in *elastic.in* as much as possible, and then set the parameters specific to the elastic calculation:

- `task` : Set the calculation type; this calculation is an elastic calculation.
- `elastic.dispOrder` : Set the method of atomic vibration for elastic calculations; 1 corresponds to the central difference method;
- `elastic.dispRange` : Set the magnitude of atomic displacement for elastic calculations;

*structure.as* The file is referenced as follows:

```
Total number of atoms
8
Lattice
5.43070000 0.00000000 0.00000000
0.00000000 5.43070000 0.00000000
0.00000000 0.00000000 5.43070000
Cartesian
Si 0.67883750 0.67883750 0.67883750
Si 3.39418750 3.39418750 0.67883750
Si 3.39418750 0.67883750 3.39418750
Si 0.67883750 3.39418750 3.39418750
Si 2.03651250 2.03651250 2.03651250
Si 4.75186250 4.75186250 2.03651250
Si 4.75186250 2.03651250 4.75186250
Si 2.03651250 4.75186250 4.75186250
```

**Note**

1. **When performing elastic calculations, the convergence accuracy of self-consistent calculations should be increased. It is recommended to set it to 1.0e-6 or higher.**
2. **Fixed atoms are not supported in elastic calculations.**

### 2.14.2 run program execution

After preparing the input files, upload the *:guilabel: `elastic.in* and *:guilabel: `structure.as* files to the server and run *:guilabel: `DS-PAW elastic.in* following the method described in Structure Relaxation.

### 2.14.3 Analysis of the calculation results.

After the calculation based on the input files mentioned above, the following three files will be generated: *DS-PAW.log*, *elastic.h5*, and *elastic.txt*.

- *DS-PAW.log*: Log file generated after the DS-PAW elastic calculation;
- *elastic.h5* : **h5** data file generated after the elasticity calculation. The elastic modulus is stored in *elastic.h5*. For detailed data structure, please refer to *Output File Format Specification*;
- *elastic.txt* : A **txt** text file generated after the elasticity calculation. This file contains elasticity-related data, consistent with the *elastic.h5* file, for easy user access.

The elastic constant matrix obtained from the *elastic.txt* file is as follows:

**Stiffness Elasticity Matrix:**

| | | | | | |
|---|---|---|---|---|---|
| 158.7644 | 62.9858 | 62.9858 | 0.0000 | -0.0000 | 0.0000 |
| 62.9858 | 158.7644 | 62.9858 | 0.0000 | 0.0000 | 0.0000 |
| 62.9858 | 62.9858 | 158.7644 | -0.0000 | 0.0000 | 0.0000 |
| 0.0000 | 0.0000 | -0.0000 | 75.8807 | -0.0000 | 0.0000 |
| -0.0000 | 0.0000 | 0.0000 | -0.0000 | 75.8807 | -0.0000 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | -0.0000 | 75.8807 |

**Flexibility Elasticity Matrix:**

| | | | | | |
|---|---|---|---|---|---|
| 0.0081 | -0.0023 | -0.0023 | -0.0000 | 0.0000 | -0.0000 |
| -0.0023 | 0.0081 | -0.0023 | -0.0000 | -0.0000 | 0.0000 |
| -0.0023 | -0.0023 | 0.0081 | 0.0000 | -0.0000 | 0.0000 |
| -0.0000 | -0.0000 | 0.0000 | 0.0132 | 0.0000 | -0.0000 |
| 0.0000 | -0.0000 | -0.0000 | 0.0000 | 0.0132 | 0.0000 |
| -0.0000 | 0.0000 | 0.0000 | -0.0000 | 0.0000 | 0.0132 |

**Bulk Modulus, Shear Modulus, Youngs Modulus, and Poissons Ratio:**

| Properties | Vogit | Reuss | Hill |
|---|---|---|---|
| BulkModulus(GPa) | 94.9120 | 94.9120 | 94.9120 |
| ShearModulus(GPa) | 64.6841 | 61.5016 | 63.0929 |
| YoungModulus(GPa) | 158.1297 | 151.7315 | 154.9452 |
| PoissonRatio | 0.2223 | 0.2336 | 0.2279 |

The Si system is cubic. This crystal system has three independent matrix elements: **C11**, **C12**, and **C44**, corresponding to 158.7644, 62.9858, and 75.8807 in the table, respectively.

## 2.15 NEB Transition State Calculation

This section introduces how to perform transition state calculations (CI-NEB) in DS-PAW using the example of H diffusion on the Pt(100) surface, and how to analyze the results graphically.

### 2.15.1 Transition state calculation input file $Pt$

The input files include a parameter file, *neb.in*, and multiple structure files, *structureNo.as*. The *neb.in* file is as follows:

```
task = neb

sys.structure = structure.as
sys.functional  = PBE
sys.spin = none
sys.symmetry = true

cal.ksamping = G
cal.kpoints = [3,3,1]
cal.cutoffFactor = 1.0
cal.smearing = 1
cal.sigma = 0.05

neb.freedom = atom
neb.springK = 5
neb.images = 3
neb.iniFin = true
neb.method = LBFGS
neb.convergence = 0.03
neb.stepRange = 0.1
neb.max = 60

```

(continues on next page)

(continued from previous page)

```
io.wave = false
io.charge = false
```

*neb.in* Input Parameters:

In the transition state calculation, you can try to keep the parameters of sys. and cal. in *neb.in*, and then set the parameters specific to the transition state calculation.

- `task` : Sets the calculation type; in this case, its a NEB transition state calculation.
- `neb.stepRange`: Sets the step size for structure relaxation in the NEB transition state calculation;
- `neb.max` : Sets the maximum number of steps for structure relaxation in the NEB calculation;
- `neb.iniFin` : Controls whether self-consistent calculations are performed for the initial and final structures in the transition state calculation; true means self-consistent calculations are performed.
- `neb.springK` : Sets the spring constant K in the transition state calculation;
- `neb.images` : Set the number of intermediate images in the NEB calculation;
- `neb.method` : Sets the algorithm used for the transition state calculation;
- `neb.convergence` : Sets the force convergence criterion for the nudged elastic band (NEB) transition state calculation;

*structure.as* is required to provide multiple, and the initial state structure *structure00.as* is referenced as follows

```
Total number of atoms
13
Lattice
5.60580000 0.00000000 0.00000000
0.00000000 5.60580000 0.00000000
0.00000000 0.00000000 16.81740000
Cartesian Fix_x Fix_y Fix_z
H 2.80881670 4.20393628 6.94088012 F F F
Pt 1.40145000 1.40145000 1.98192999 T T T
Pt 4.20434996 1.40145000 1.98192999 T T T
Pt 1.40145000 4.20434996 1.98192999 T T T
Pt 4.20434996 4.20434996 1.98192999 T T T
Pt 0.00272621 0.00056545 3.91746017 F F F
Pt 0.00271751 2.80233938 3.91708172 F F F
Pt 2.80568712 -0.00141176 3.91894328 F F F
Pt 2.80548220 2.80426217 3.91792247 F F F
Pt 1.39865124 1.40124680 5.84694340 F F F
Pt 4.21951864 1.40156999 5.84719575 F F F
Pt 1.38647954 4.20437926 5.89984296 F F F
Pt 4.23154392 4.20414605 5.89983612 F F F
```

Final state structure: *structure04.as* referenced as follows.

```
Total number of atoms
13
Lattice
5.60580000 0.00000000 0.00000000
0.00000000 5.60580000 0.00000000
0.00000000 0.00000000 16.81740000
```

(continues on next page)

(continued from previous page)

```
Cartesian Fix_x Fix_y Fix_z
H 1.52157824 2.80289997 6.91583941 F F F
Pt 1.40145000 1.40145000 1.98192999 T T T
Pt 4.20434997 1.40145000 1.98192999 T T T
Pt 1.40145000 4.20434997 1.98192999 T T T
Pt 4.20434997 4.20434997 1.98192999 T T T
Pt 0.02556963 0.00000000 3.90765450 F F F
Pt 0.02708862 2.80290000 3.91082177 F F F
Pt 2.83159105 0.00000000 3.91547525 F F F
Pt 2.82981856 2.80290000 3.90913282 F F F
Pt 1.45998966 1.38039927 5.88134827 F F F
Pt 4.25691060 1.38811299 5.84551487 F F F
Pt 1.45998966 4.22540069 5.88134827 F F F
Pt 4.25691060 4.21768697 5.84551487 F F F
```

**Note**

1. **Structure relaxation is required for the initial and final states before NEB calculation.**
2. **The generation of intermediate structures can be done by calling the `neb_interpolate_structures.py` script in Tutorial for Auxiliary Tools - Transition State Section. After interpolation, the `neb_visualize.py` script can be called to preview the interpolated structures, and the `calc_dist.py` script can be called to check if the distances between images are reasonable.**
3. **For transition state calculations, the structure file structureNo.as needs to be placed in a folder named No, where the folder number corresponds to the structure file number. A neb.in file should be placed outside the folders. Run the DS-PAW program in the directory where neb.in is located.**
4. The number of cores used when executing the transition state calculation should be set to an integer multiple of the number of images.

### 2.15.2 run program execution

Once the input file is ready, upload the *neb.in* file and folders containing *structureNo.as* files to the server and run the *DS-PAW neb.in* as described in structure relaxation.

### 2.15.3 Analysis of calculation results

After the calculation is completed based on the input files described above:

The folders containing the initial and final state structures will generate output files such as *DS-PAW.log*, *latestStructure00.as*, and *scf.h5* from the self-consistent field (SCF) calculations.

Intermediate structure *structureNo.as* folders `No` (folders containing intermediate structures for transition state calculations, with the number of intermediate structures determined by the `neb.images` parameter) will generate output files such as *nebNo.h5* and *latestStructureNo.as* from the structure optimization.

The outermost directory will generate the files *DS-PAW.log* and *neb.h5*, where *neb.h5* is a summary of the information in the *nebNo.h5* files under the `No` folders.

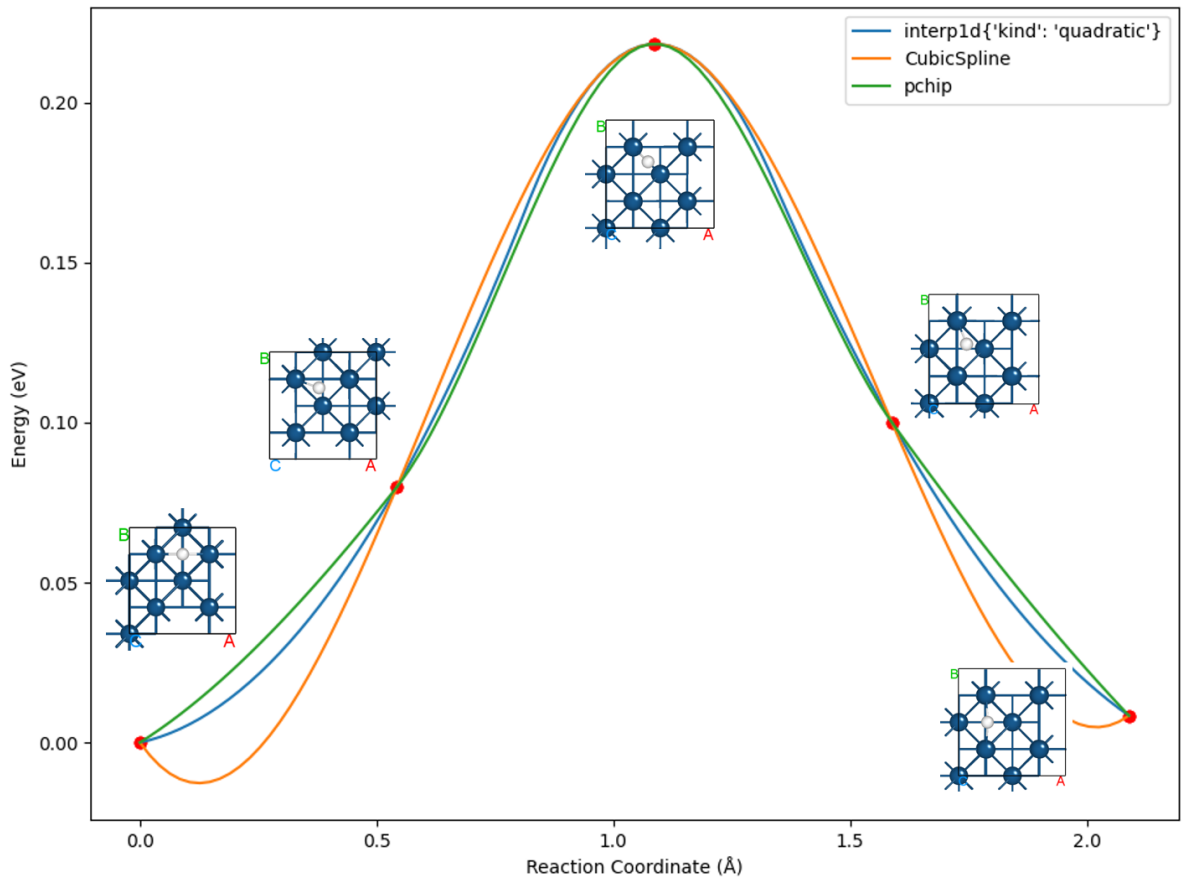
- *DS-PAW.log* : Log file obtained after DS-PAW transition state calculation;
- *neb.h5* : The **h5** data file after the transition state calculation is completed; the reaction coordinates and energy changes, etc., are saved in *neb.h5*. For the specific data structure, please refer to the *Output File Format Specification* section;

You can use the **python** script `8neb_check_results.py` to analyze the results of the NEB calculation. The analysis script should be executed in the complete NEB calculation directory. See the *Auxiliary Tool User Guide* section for specific instructions.

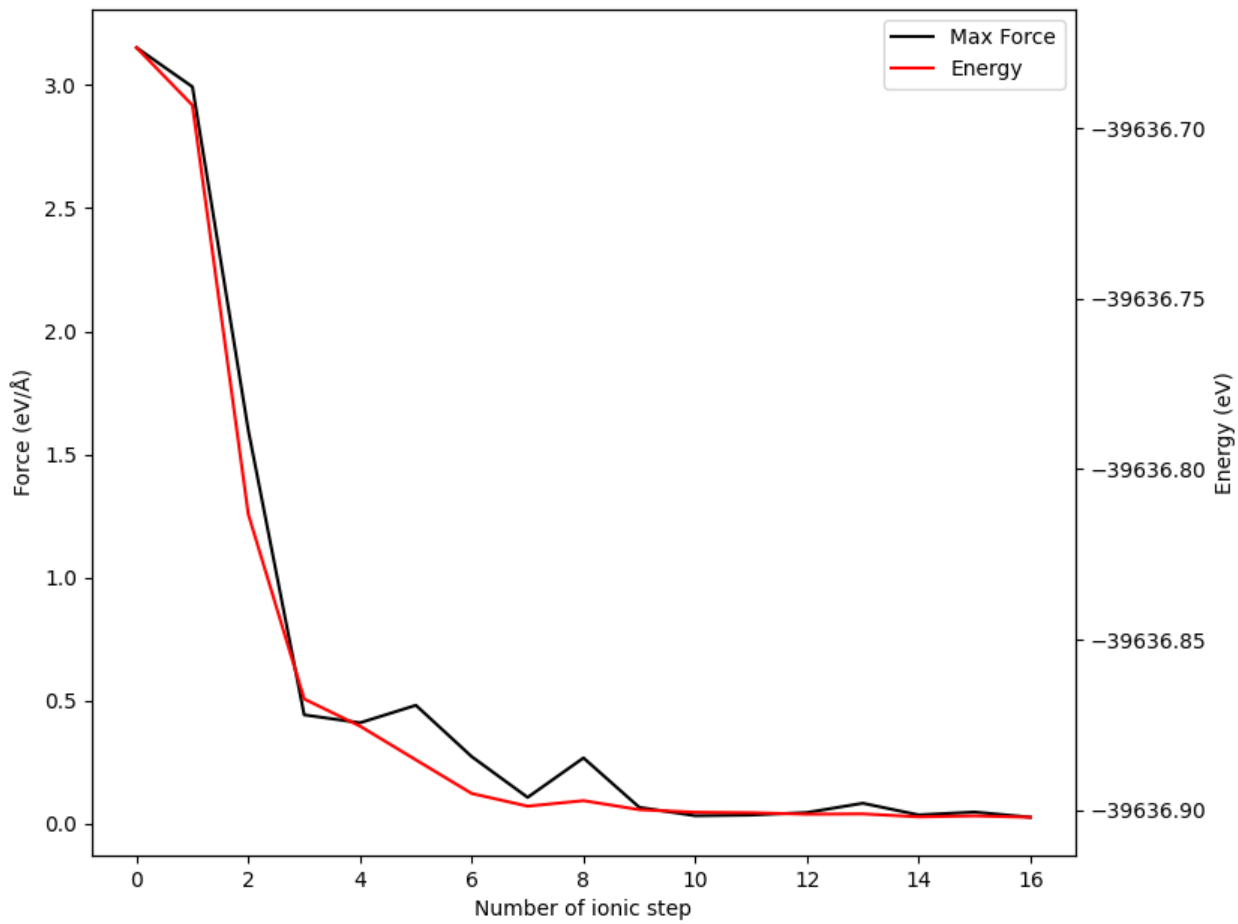
Processing yields tables of energies and forces for each NEB configuration:

| Image | Force (eV/Å) | Reaction coordinate (Å) | Energy (eV) | Delta energy (eV) |
|---|---|---|---|---|
| 00 | 0.1803 | 0.0000 | -39637.0984 | 0.0000 |
| 01 | 0.0263 | 0.5428 | -39637.0186 | 0.0798 |
| 02 | 0.0248 | 1.0868 | -39636.8801 | 0.2183 |
| 03 | 0.2344 | 1.5884 | -39636.9984 | 0.1000 |
| 04 | 0.0141 | 2.0892 | -39637.0900 | 0.0084 |

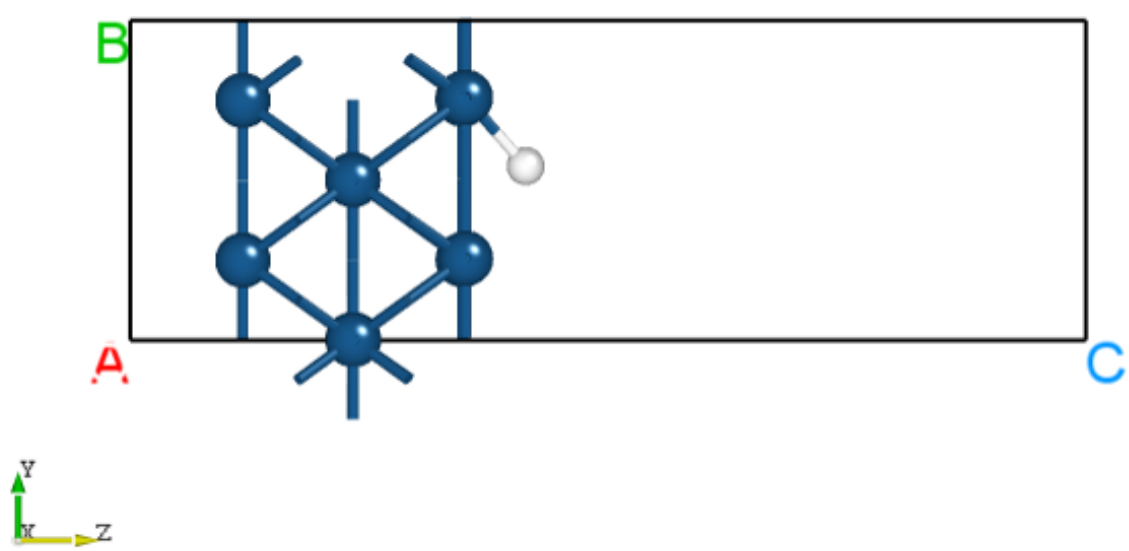
The resulting barrier curve effect should look like this:




The energy and force of the 02 image obtained during the relaxation process are shown as follows:




Alternatively, you can use the **python** script *neb_movie.py* to analyze the trajectory changes in the transition state search. The generated *neb_movie.json* file can be opened with Device Studio, and a frame is captured as shown below:



## 2.16 Phonon Dispersion Calculation

This section introduces how the DS-PAW code performs phonon calculations and computes phonon band structures and phonon density of states (DOS). DS-PAW supports two methods for phonon spectrum calculations: the finite displacement (fd) method and the density functional perturbation theory (DFPT) method. Taking a single MgO system as an example, this section explains how to calculate phonon bands and DOS using both methods, and analyzes the phonon band structure and DOS plots.

### 2.16.1 $MgO$ Phonon Dispersion Calculation Input File

The input files consist of the parameter file *phonon.in* and the structure file *structure.as*. The *phonon.in* file is as follows:

```
task = phonon

sys.structure = structure.as
sys.functional = PBE
sys.spin = none

cal.methods = 1
cal.smearing = 1
sys.symmetry = true
scf.convergence = 1.0e-07
cal.ksamping = G
cal.kpoints = [3,3,3]
cal.sigma = 0.25

phonon.type = bandDos
phonon.structureSize = [2,2,2]
phonon.primitiveUVW = [0.0, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.0]
phonon.method = dfpt
phonon.qpoints = [41,41,41]
phonon.dosRange = [0,20]
phonon.qpointsLabel = [G,X,W,G,M]
phonon.qpointsCoord = [0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.5]
phonon.qpointsNumber = 51

io.charge = false
io.wave = false
```

*phonon.in* input parameter introduction:

In phonon calculations, you can generally retain the parameters of *sys.* and *cal.* in *:guilabel:`phonon.in* and then set the specific parameters for phonon calculations:

- `task` : Sets the calculation type, which is phonon for this calculation;
- `phonon.type` : Set the type of phonon calculation, bandDos corresponds to calculating phonon band structure and density of states;
- `phonon.structureSize` : Set the size of the supercell for phonon calculations;
- `phonon.primitiveUVW` : Set the coefficients of the primitive cell UVW for phonon band calculations;
- `phonon.method` : Sets the method for phonon calculations, with dfpt indicating the Density Functional Perturbation Theory method;

- `phonon.qpoints` : Set the q-space grid sampling for phonon calculation to 41*41*41;
- `phonon.dosRange` : Sets the energy range for phonon density of states calculation to [0, 20];
- `phonon.qpointsLabel` : Set the labels for high-symmetry points in phonon band calculations;
- `phonon.qpointsCoord` : Set the coordinates of high-symmetry points for phonon band calculations.
- `phonon.qpointsNumber` : Set the interval between adjacent high-symmetry points for phonon band calculations;

The *structure.as* file is referenced as follows:

```
Total number of atoms
8
Lattice
   4.2555564654942897    0.0000000000000000    0.0000000000000000
   0.0000000000000000    4.2555564654942888    0.0000000000000000
   0.0000000000000000    0.0000000000000000    4.2555564654942897
Direct
Mg  0.0000000000000000  0.0000000000000000  0.0000000000000000
Mg  0.0000000000000000  0.5000000000000000  0.5000000000000000
Mg  0.5000000000000000  0.0000000000000000  0.5000000000000000
Mg  0.5000000000000000  0.5000000000000000  0.0000000000000000
O   0.5000000000000000  0.5000000000000000  0.5000000000000000
O   0.5000000000000000  0.0000000000000000  0.0000000000000000
O   0.0000000000000000  0.5000000000000000  0.0000000000000000
O   0.0000000000000000  0.0000000000000000  0.5000000000000000
```

**Note**

1. **When performing phonon calculations, the convergence accuracy of the self-consistent calculation should be increased; it is recommended to set it to 1.0e-7 or higher.**
2. **When performing phonon calculations with symmetry enabled, it is recommended to increase the accuracy of symmetry determination appropriately. The parameter sys.symmetryAccuracy can be set to 1.0e-6 or smaller to help obtain accurate calculation results.**
3. **phonon.iniPhonon can specify the path to read the phonon calculation (phonon.type = phonon) generated phonon.h5 file, enabling direct calculation of band structures and density of states.**
4. **phonon.type controls the type of phonon calculation. phonon corresponds to phonon calculation, band corresponds to phonon band calculation, dos corresponds to phonon density of states calculation, and bandDos corresponds to simultaneous calculation of phonon band and density of states. When phonon.type = band/dos/bandDos and no file path is specified for phonon.iniPhonon, the program first automatically performs the phonon calculation for phonon.type = phonon, and then calculates the band structure or density of states according to the task.**

### 2.16.2 run program execution

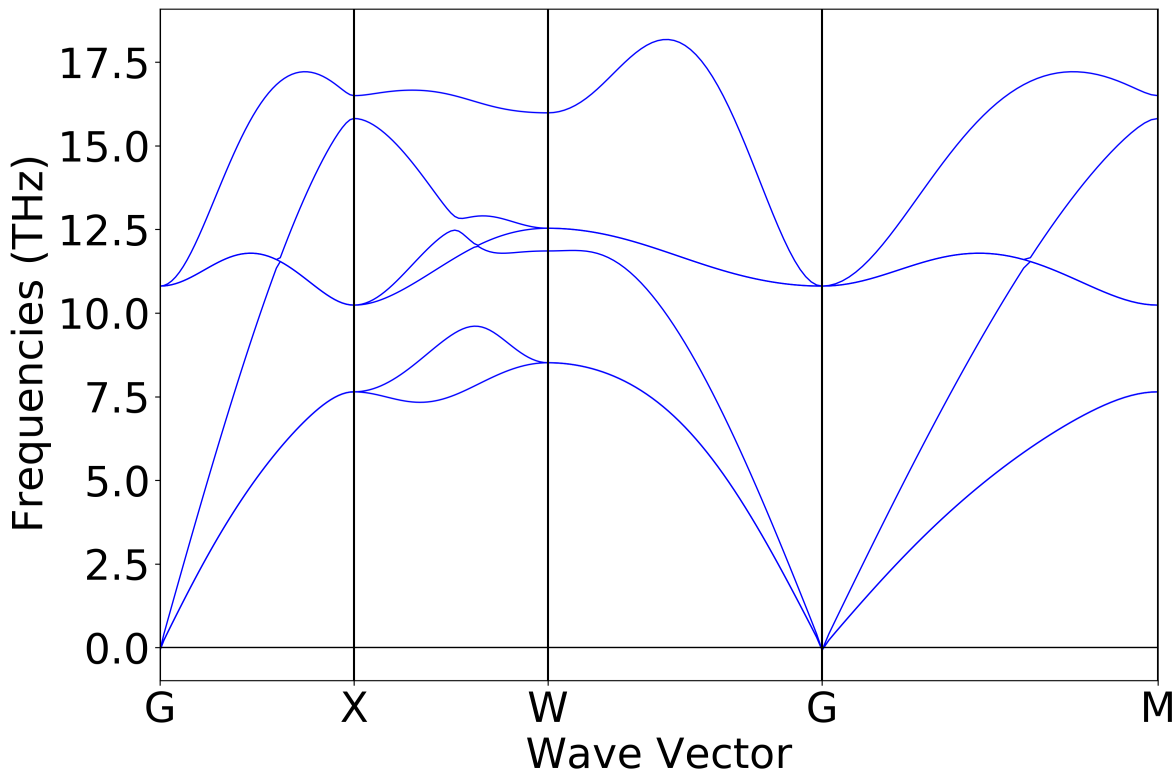
After preparing the input files, upload the *phonon.in* and *structure.as* files to the server and run them, executing *DS-PAW phonon.in* as described in Structure Relaxation.

### 2.16.3 Analysis of Calculation Results

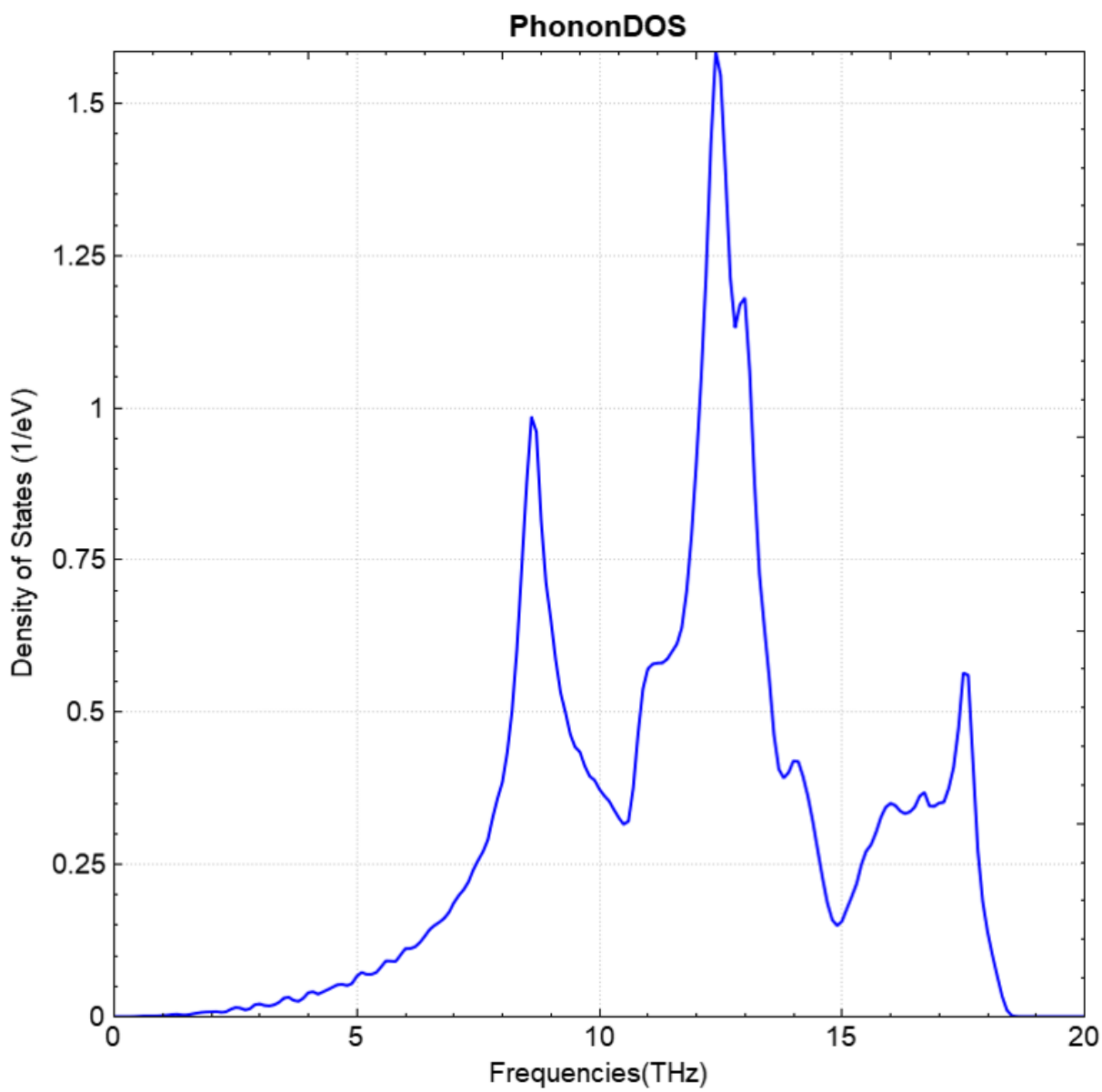
Based on the input files mentioned above, the calculation will generate the following output files: *DS-PAW.log*, *phonon.h5*, *dfpt.json*, and *dfpt.as*.

- *DS-PAW.log* : The log file generated after the DS-PAW phonon calculation.
- *dfpt.as* : Supercell structure file for phonon calculations, and this file is read during phonon calculations.
- *dfpt.json* : Parameter file for phonon calculations, which is consistent with the information in the *phonon.in* file. This file is read when calculating phonons.
- *phonon.h5* : The **h5** data file after the phonon calculation is completed; the phonon band data is stored in *phonon.h5* at this point, and the specific data structure is detailed in the *Output File Format Specification* section;

You can use a **python** script to process the data in *phonon.h5*. The phonon band structure and density of states plots obtained after processing should look like (a) and (b) below:




(a)



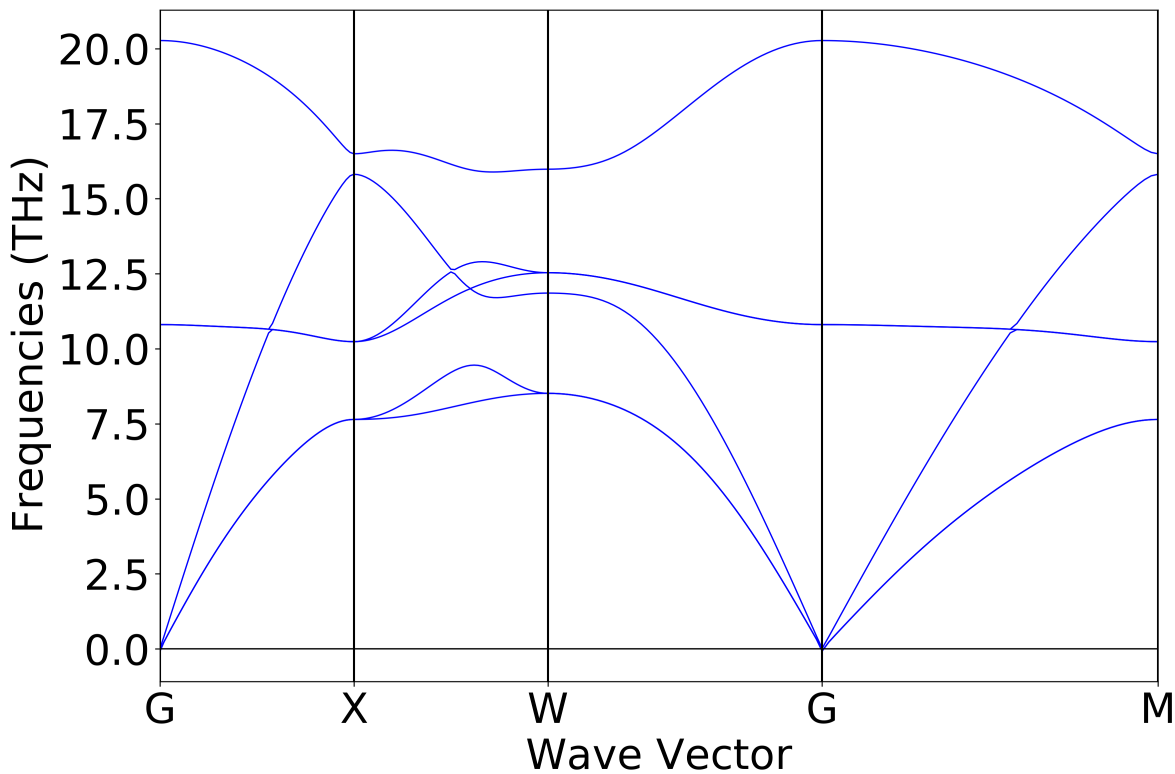

(b)

### 2.16.4 Analysis of NAC calculation results

The previous section presented the phonon band calculation without considering long-range interactions. To perform phonon calculations with the **non-analytical term correction (nac)**, you can add the following two parameters to the *phonon.in* file shown in the previous section:

```
phonon.dfptEpsilon=true
phonon.nac = true
```

The resulting phonon band structure should look like (c) below:




(c)

### 2.16.5 fdphonon: Finite Displacement Method for Phonon Calculation

The input file for **finite displacement (fd)** phonon calculations is as follows. Simply modify the parameter `phonon.method = dfpt` to `phonon.method = fd`. Note that the output files generated by the fd method are different from those generated by the dfpt method.

```
task = phonon

sys.structure = structure.as
sys.functional = PBE
sys.spin = none

cal.methods = 1
cal.smearing = 1
sys.symmetry = true
scf.convergence = 1.0e-07
cal.ksamping = G
cal.kpoints = [3,3,3]
cal.sigma = 0.25

phonon.type = bandDos
phonon.structureSize = [2,2,2]
phonon.primitiveUVW = [0.0, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.0]
phonon.method = fd
phonon.qpoints = [41,41,41]
```

(continues on next page)

(continued from previous page)

```
phonon.qpointsLabel = [G,X,W,G,M]
phonon.qpointsCoord = [0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5,␣
↪0.5, 0.5]
phonon.qpointsNumber = 51

io.charge = false
io.wave = false
```

For the MgO system, for example, when `phonon.structureSize` is set to `[2,2,2]`, after the finite difference (FD) calculation is completed, two files, *DS-PAW.log* and *phonon.h5*, will be generated, along with folders `001` and `002`. Folder `001` contains the files *input.json* and *disp-001.as*, and folder `002` contains *input.json* and *disp-002.as*. The two files in each subfolder are equivalent to the **in** file (input parameters) and the **as** file (structure parameters). The number of generated folders (001, 002, ) depends on the symmetry of the system.

Using a **python** script to process the *phonon.h5* file obtained from the finite displacement method calculation, the resulting band structure and density of states plots are consistent with plots (a) and (b) calculated using the dfpt method.

**Note**

1. **The calculation of dielectric constant is only possible when phonon.method = dfpt.**
2. **The switch of phonon.nac is only effective when phonon.method = dfpt and phonon.dfptEpsilon=true**

## 2.17 soc spin-orbit coupling calculation

This section describes how DS-PAW performs spin-orbit coupling calculations. Taking the $Bi_2Se_3$ system as an example, we use a two-step method to calculate and analyze the band structure.

### 2.17.1 $Bi_2Se_3$ Spin-Orbit Coupling Calculation Input File

First, a self-consistent calculation is performed: the input file contains the parameter file *soi.in* and the structure file *structure.as*, and *soi.in* is as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = false
sys.functional = PBE
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [7, 7, 7]
cal.cutoffFactor = 1.5
#soi related
sys.spin= non-collinear
sys.soi = true
#outputs
io.charge = true
io.wave = false
```

*soi.in* Input Parameter Description:

In spin-orbit coupling calculations, you can generally retain the parameters from *sys.* and *cal.* in the *:guilabel:`soi.in* file and then configure the specific parameters for the spin-orbit coupling calculation:

- `sys.spin` : Sets the spin type of the system; non-collinear means non-collinear spin.
- `sys.soi` : Controls whether to consider spin-orbit coupling effect; this parameter is effective when sys.spin=non-collinear;

The following describes the *structure.as* file:

```
Total number of atoms
5
Lattice
-2.069  -3.583614  0.000000
2.069  -3.583614  0.000000
0.000   2.389075  9.546667
Direct
Bi 0.3990 0.3990 0.6970
Bi 0.6010 0.6010 0.3030
Se 0.0000 0.0000 0.5000
Se 0.2060 0.2060 0.1180
Se 0.7940 0.7940 0.8820
```

Input file for band structure calculation: *soiband.in*, content as follows

```
# task type
task = band
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
#scf related
cal.methods = 2
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [7, 7, 7]
cal.cutoffFactor = 1.5
#band related
cal.iniCharge = ./rho.bin
band.kpointsCoord = [0.00000000,0.00000000,0.00000000,0.00000000,0.00000000,0.50000000,0.50000000,0.50000000,0.00000000,0.00000000,0.00000000,0.00000000,0.50000000,0.00000000,0.00000000]
band.kpointsLabel = [G,Z,F,G,L]
band.kpointsNumber = [20,20,20,20]
band.project = true
#soi related
sys.spin= non-collinear
sys.soi = true
```

Introduction to input parameters in *soiband.in*:

In spin-orbit coupling band calculations, the parameters from the self-consistent calculation and spin-orbit coupling calculation are retained in *soiband.in*. After that, you can set the specific parameters for the band calculation.

**Note**

1. **Initial magnetic moment setup refers to Application Case - Antiferromagnetic Calculation of the NiO System. Set the Mag tag on the seventh line of the structure.as file.**

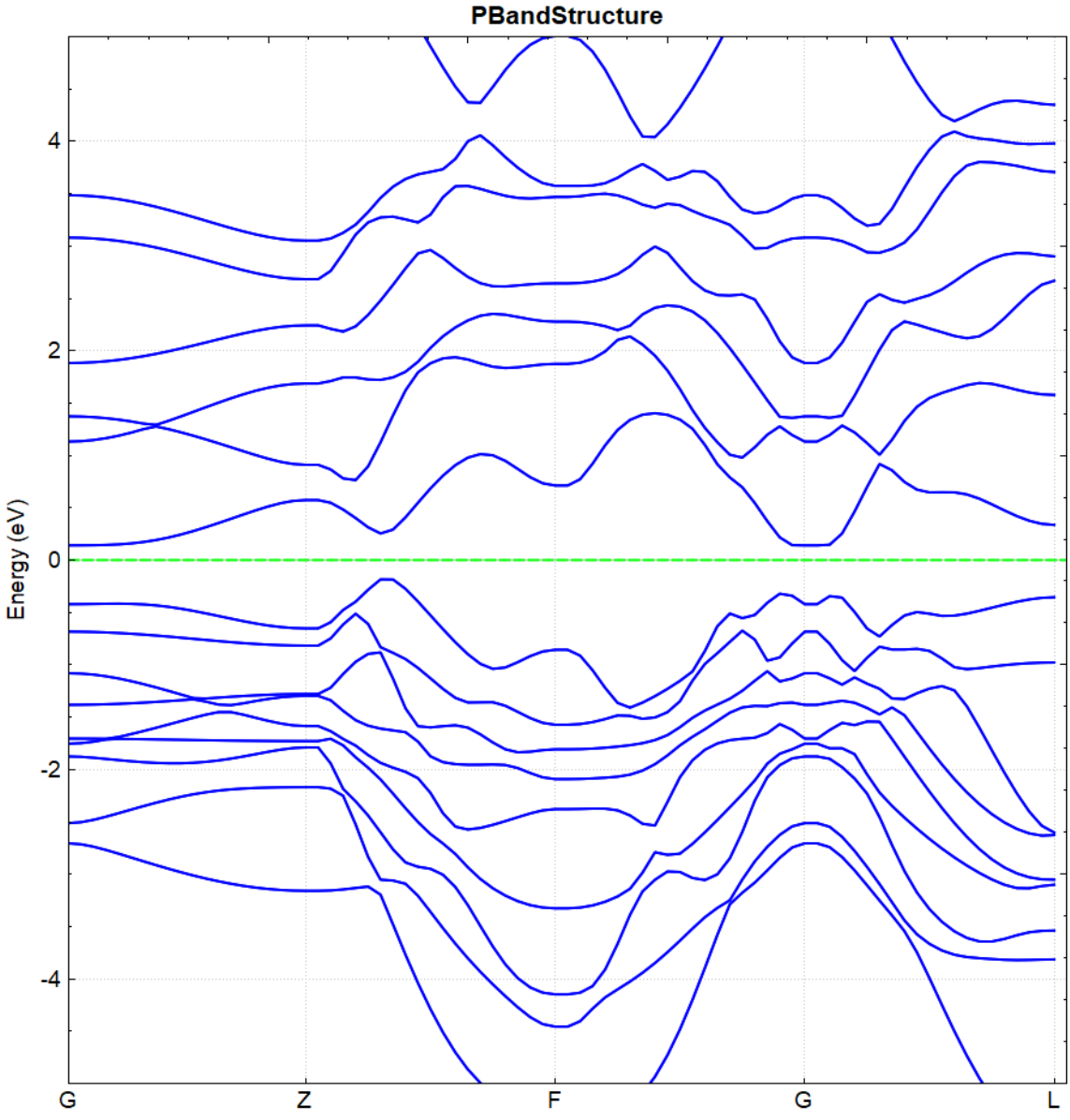
### 2.17.2 Run the program

After preparing the input files, upload the *soi.in*, *soiband.in*, and *structure.as* files to the server for execution. Run *DS-PAW soi.in* and *DS-PAW soiband.in* separately, following the methods described in the structural relaxation section.

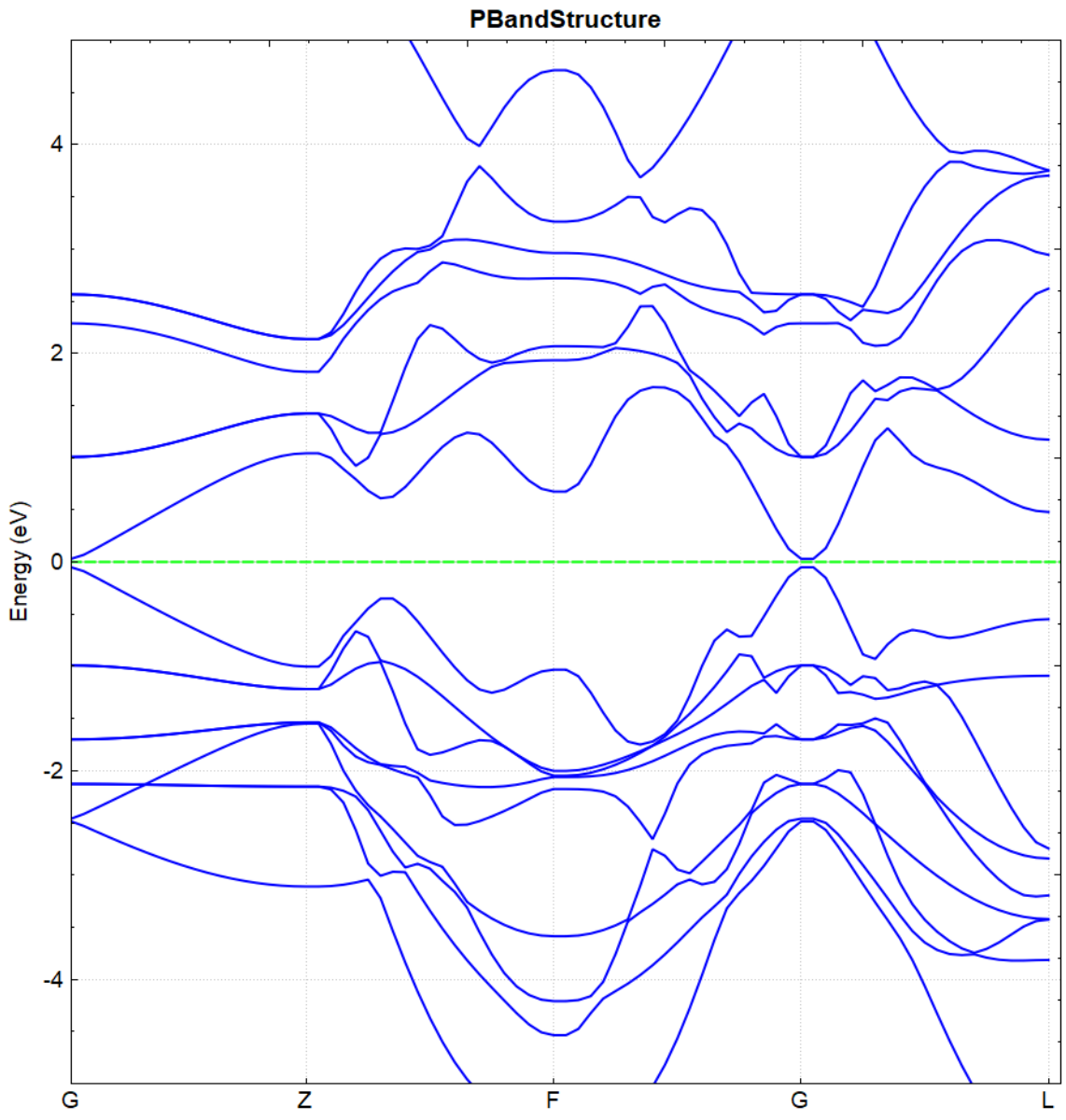
### 2.17.3 Analysis Result Analysis

Based on the input files, after calculation completion, output files such as *DS-PAW.log*, *scf.h5*, and *band.h5* will be generated.

The processing of *band.h5* follows the same method as the band calculation described in Section 2.3. The resulting band structure should be as shown in Figure (a) below. Additionally, a calculation without spin-orbit coupling should yield the band structure shown in Figure (b) below:




(a)




(b)

The **BandGap** values are read from *DS-PAW.log*. The band gap values for Figures (a) and (b) are **0.3251** eV and **0.0814** eV, respectively. This leads to the conclusion that spin-orbit coupling calculations increase the band gap between the valence and conduction bands.

## 2.18 AIMD molecular dynamics simulation

This section will introduce how to perform molecular dynamics simulations in DS-PAW, using a water molecule system as an example.

### 2.18.1 Input file for $H_2O$ molecular dynamics simulation

The input files include the parameter file *aimd.in* and the structure file *structure.as*. *aimd.in* is shown below:

```
#task type
task = aimd

#system related
sys.structure = structure.as
sys.symmetry = false
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [1, 1, 1]
cal.sigma = 0.1

#aimd related
aimd.ensemble = NPT
aimd.thermostat = langevin
aimd.atomFCoeffElements = [H_1]
aimd.atomFCoeffs = [1]
aimd.latticeFCoeff = 1
aimd.pressure = 100
aimd.timeStep = 1
aimd.totalSteps = 2000
aimd.iniTemp = 2000

#outputs
io.charge = false
io.wave = false
```

*aimd.in* Input Parameters:

In the molecular dynamics simulation calculation, try to keep the parameters in sys. and cal. in *aimd.in*, then set the parameters specific to the molecular dynamics simulation calculation.

- `task` : Sets the calculation type. In this case, the calculation is an AIMD molecular dynamics simulation.
- `aimd.ensemble` : Specifies the ensemble used for the molecular dynamics simulation. In this case, the ensemble is set to NPT.
- `aimd.thermostat`: Sets the thermostat or barostat used in the molecular dynamics simulation. In this example, the Langevin thermostat and barostat are used.
- `aimd.atomFCoeffElements` : Specifies the element names of the atoms considered as Langevin atoms. In this example, one hydrogen atom is set as a Langevin atom, and it is renamed to H_1;
- `aimd.atomFCoeffs` : Sets the friction coefficients for atoms considered as Langevin atoms, in units of ps-1;
- `aimd.latticeFCoeff` : Sets the friction coefficient of the lattice in the Langevin thermostat, unit ps-1;
- `aimd.pressure` : Sets the target pressure value for NPT simulations, in kbar;
- `aimd.timeStep` : Sets the time step for molecular dynamics simulation, in fs;
- `aimd.totalSteps` : Set the total number of steps for the molecular dynamics simulation;

- `aimd.iniTemp` : Sets the initial temperature for molecular dynamics simulations, in K;

The *structure.as* file is referenced as follows:

```
Total number of atoms
3
Lattice
4.00000000 0.00000000 0.00000000
0.00000000 4.00000000 0.00000000
0.00000000 0.00000000 4.00000000
Cartesian
H 2.63934013 1.89542007 1.58223984
H_1 1.36065987 2.11498988 2.45934006
O 1.65002999 1.88501012 1.54065994
```

**Note**

1. **The element renaming rule is original element name + underscore + custom field.**
2. **In this example, the second hydrogen atom is set as a Langevin atom and renamed to H_1. The element name for this atom needs to be manually modified in the structure.as file.**
3. **Since a custom element name H_1 exists in the calculation system, the program will automatically search for the pseudopotential of the corresponding H element for H_1, and the user does not need to prepare a new pseudopotential.**

### 2.18.2 run the program

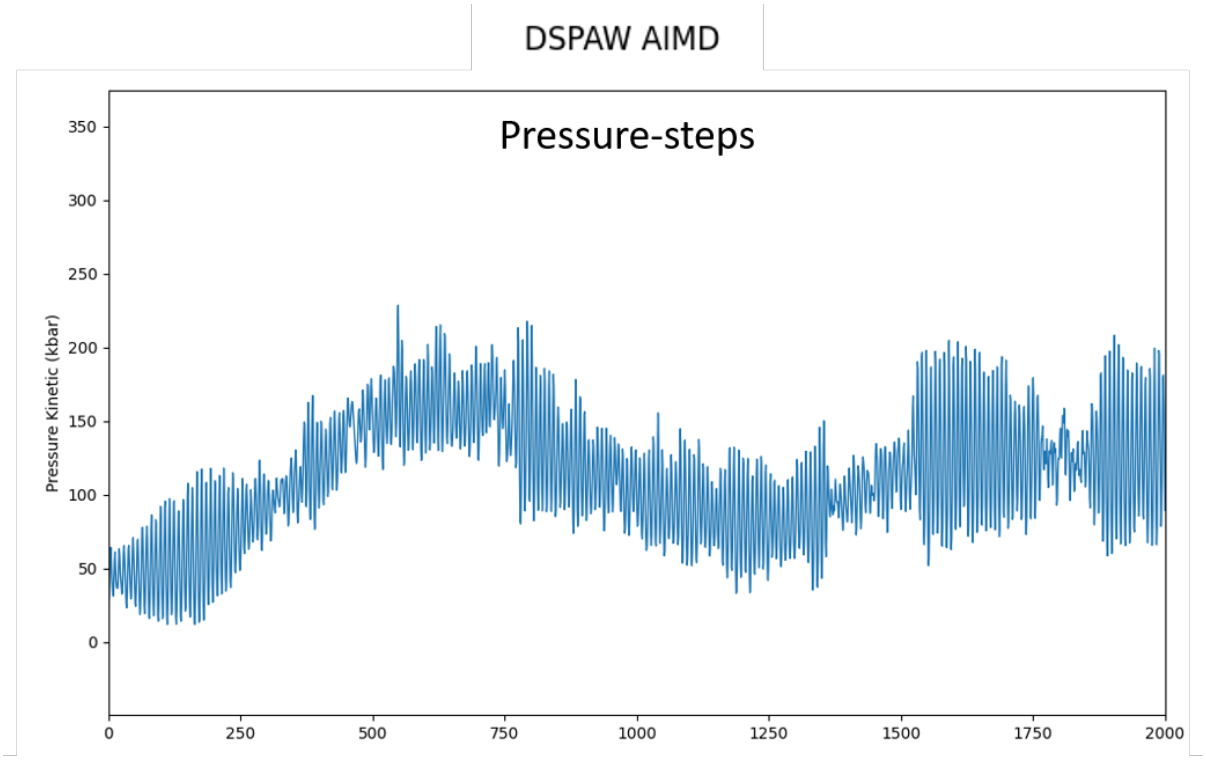
Once the input files are prepared, upload the files *aimd.in* and *structure.as* to the server and run them. Execute *DS-PAW aimd.in* as described in Structure Relaxation.

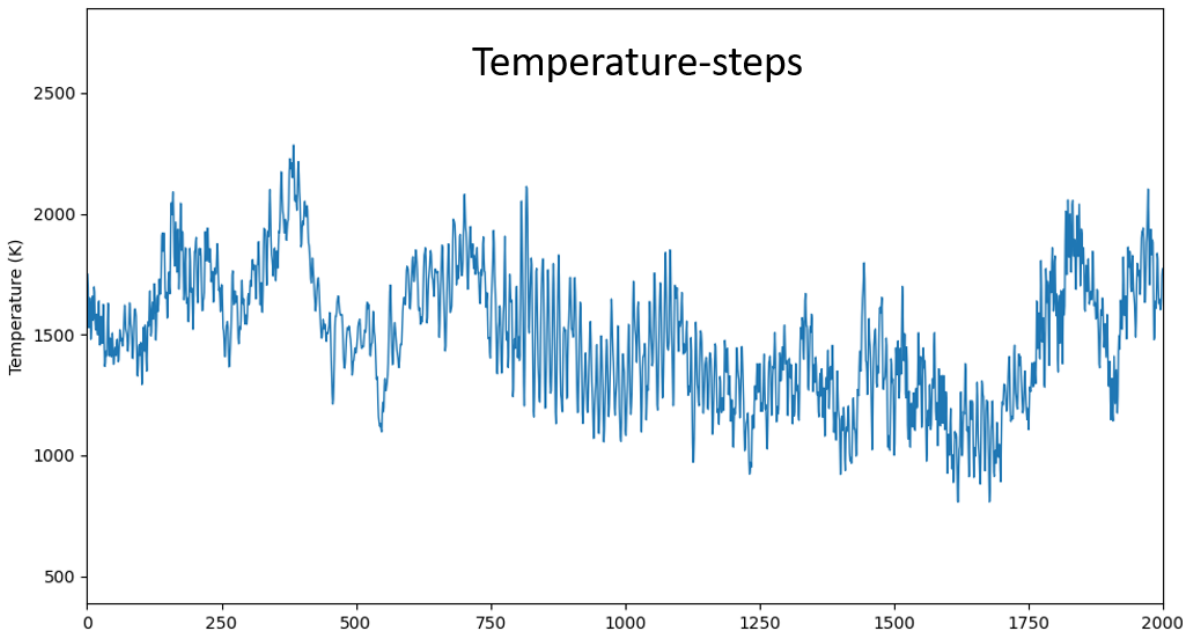
### 2.18.3 Analysis of calculation results.

Based on the input files mentioned above, the calculation will produce output files such as *DS-PAW.log*, *aimd.h5*, and *latestStructure.as* upon completion.

- *DS-PAW.log* : Log file generated by the DS-PAW molecular dynamics simulation.
- *aimd.h5* : The **h5** output file corresponding to the molecular dynamics calculation; atom positions, system energy, temperature, and other data during the simulation are saved in *aimd.h5*. For details on the data structure, see the *Output File Format Specification* section.
- *latestStructure.as* : The final state as structure file from molecular dynamics simulation, storing the final configuration and velocity information;

Data processing of the *aimd.h5* file can be performed using a **python** script, as detailed in the *Auxiliary Tool User Guide* section. The following figures should show the **pressure vs. time** and **temperature vs. time** curves obtained from a 2000-step NPT ensemble simulation:




**Note**

1. **Different ensembles correspond to different optional thermostat ranges: the NVE ensemble can choose the Andersen thermostat; the NVT ensemble can choose the Andersen, Nose-Hoover, and Langevin thermostats; the NPT and NPH ensembles can choose the Langevin thermostat.**
2. **To simulate a high-temperature annealing process, set `aimd.ensemble` to SA and set the initial and**

**final temperatures via `aimd.iniTemp` and `aimd.finTemp` respectively**.

3. **The parameter `aimd.finTemp` only takes effect during simulated annealing. For constant-temperature ensembles such as NPT and NVT, the final temperature is equal to the initial temperature.**
4. **When simulating a system containing langevin atoms, it is recommended to store the pseudopotential files corresponding to the langevin atoms in the calculation directory to avoid the program reporting error E3058 due to failure to find the pseudopotential files.**

## 2.19 efield plus applied electric field calculation

This section will use the band calculation of a silicene model as an example to demonstrate how to perform calculations with an external electric field in DS-PAW, and analyze the band gap opening before and after applying the electric field.

### 2.19.1 Input file for silicene calculation with external electric field in vacuum

The input file contains parameter file *Efield.in* and structure file *structure.as*, *Efield.in* is as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.sigma = 0.1
cal.cutoff = 520
cal.ksamping = G
cal.kpoints = [9, 9, 1]

scf.convergence = 1e-5

#outputs
io.charge = false
io.wave = false
io.band = true

corr.dipol=true
corr.dipolDirection = c
corr.dipolEfield = 0.2

band.kpointsLabel = [G,M,K,G]
band.kpointsCoord = [0.00000000,0.00000000,0.00000000,0.50000000,0.00000000,0.00000000,0.
↪33333333,0.33333333,0.00000000,0.00000000,0.00000000,0.00000000]
band.kpointsNumber = [100,100,100]
```

*Efield.in* Input Parameters:

The calculation is performed on top of a one-step band calculation with an external electric field. In addition to the basic parameters of the band calculation, the following new parameters are introduced:

- `corr.dipolEfield` : Sets the magnitude of the applied electric field. This parameter is only effective when `corr.dipol = true` and `corr.dipolDirection` is set;

The *structure.as* file is referenced as follows:

```
Total number of atoms
2
Lattice
3.860000   0.000000   0.000000
-1.930000   3.342860   0.000000
0.000000   0.000000  26.460000
Direct
Si   0.333333   0.166667   0.396825
Si   0.666758   0.833380   0.379216
```

### 2.19.2 run program execution

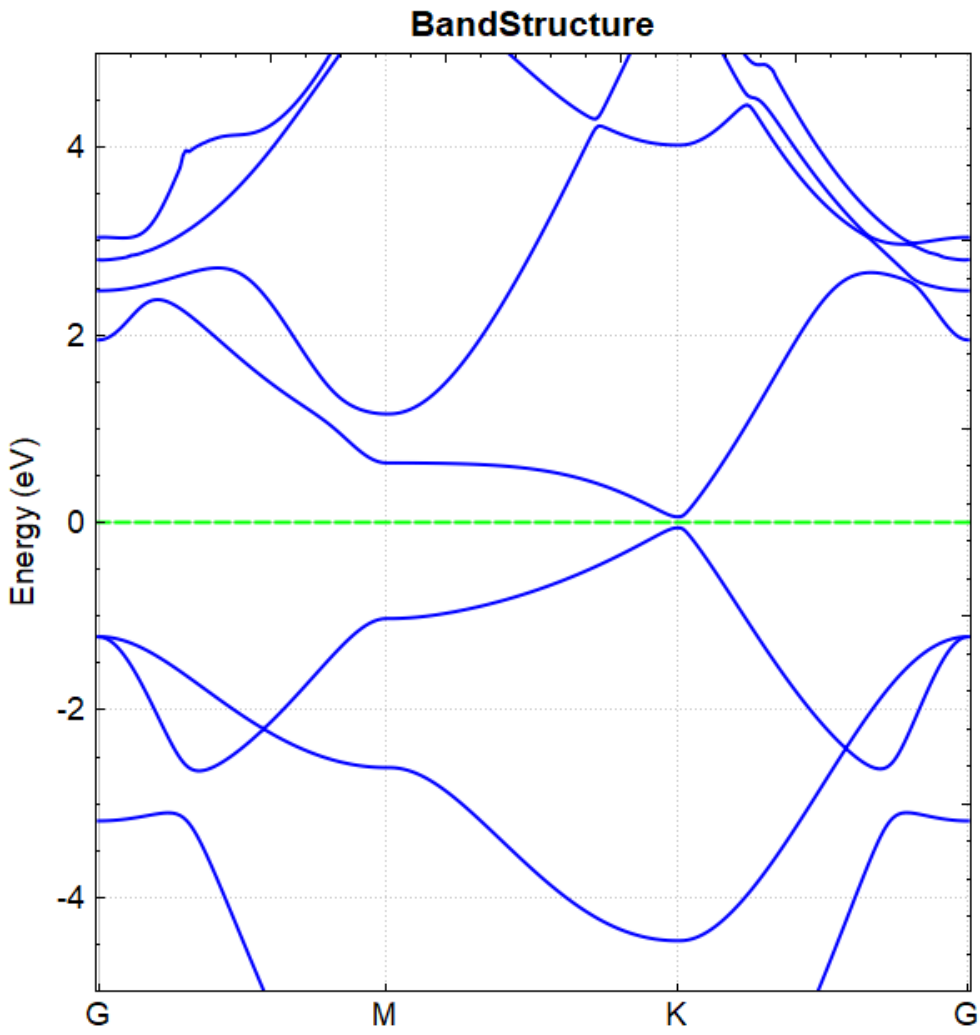
After preparing the input files, upload the *Efield.in* and *structure.as* files to the server for execution, and run *DS-PAW Efield.in* as described in the structure relaxation section.

### 2.19.3 Analysis of calculation results

Based on the input files mentioned above, the calculation will generate output files such as *DS-PAW.log* and *scf.h5*.

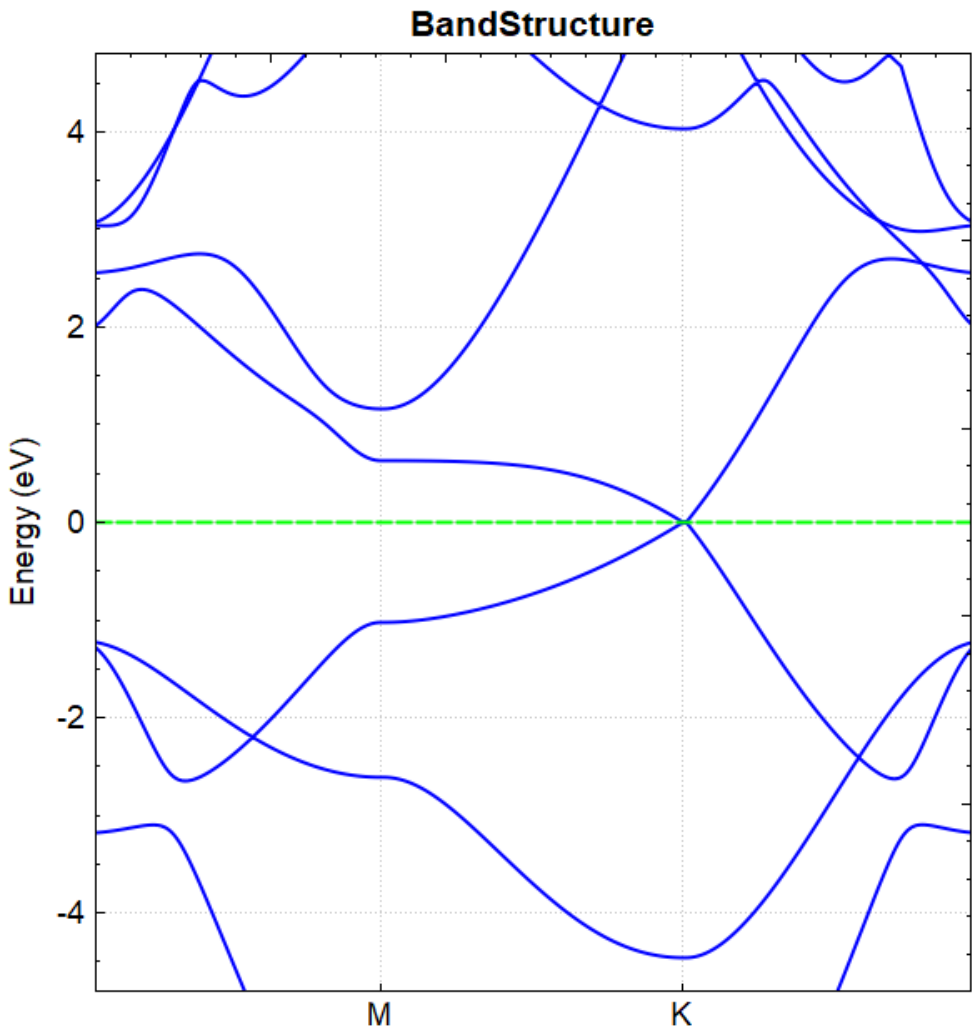
*:guilabel:`scf.h5`*: The **h5** output file corresponding to the self-consistent field (SCF) calculation. When *io.band = true*, the band structure data will be written to the *scf.h5* file;

In this example, the parameter `corr.dipolEfield = 0.2`, indicating an applied electric field strength of **0.2** eV/Å. The band structure calculated under this electric field is shown in Figure (a).




(a)

Repeating the above calculation with the parameter `corr.dipolEfield = 0`, i.e., band structure calculation without an electric field, results in the band diagram shown in Figure (b).




(b)

Comparing figures (a) and (b), we can conclude that applying an external electric field can open the band gap of silicene. The values of the **BandGap** with and without the electric field, readable from the *DS-PAW.log* file, are **0.1176** eV and **0.0010** eV, respectively.

> **Note**
>
> 1. **The unit of the external electric field, eV/Å, is also the unit of atomic force.**

## 2.20 polarization ferroelectric calculation

This section will use $HfO_2$ as an example to introduce how to perform ferroelectric calculations using modern polarization theory in DS-PAW, analyzing the ferroelectric polarization of $HfO_2$.

### 2.20.1 $HfO_2$ Ferroelectric Calculation Input File

The input files include the parameter file *polarization.in* and a series of structure files for different phases *structure.as*. The contents of *polarization.in* are as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 3
cal.smearing = 4
cal.cutoff = 520
cal.ksamping = MP
cal.kpoints = [4, 4, 4]

scf.convergence = 1e-5

#outputs
io.charge = false
io.wave = false
io.polarization = true
```

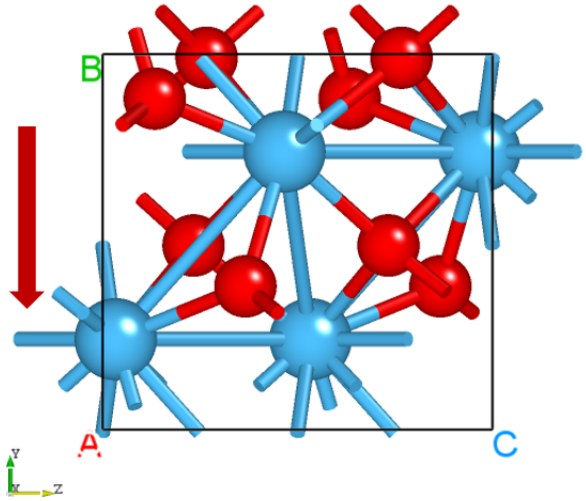
*:guilabel: `polarization.in* Input Parameters:`

This calculation performs ferroelectric calculations based on a self-consistent calculation. In addition to the basic parameters for the self-consistent calculation, the following parameters are newly added:

- `io.polarization` : Controls the switch for ferroelectric calculations in the self-consistent calculation;

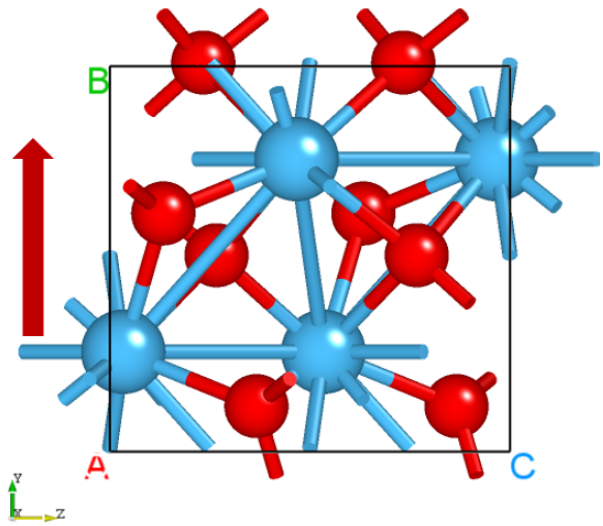
The following is a reference *structure.as* file for the ferroelectric phase structure of $HfO_2$ with polarization pointing downwards:

```
Total number of atoms
12
Lattice
5.04621935 0.00000000 0.00000000
0.00000000 5.07315250 0.00000000
0.00000000 0.00000000 5.25768906
Cartesian
Hf 1.34815269 1.22145222 0.17639072
Hf 1.34815269 3.75802848 2.45245381
Hf 3.69806665 1.22145222 2.80523525
Hf 3.69806665 3.75802848 5.08129834
O 0.35195212 1.93667284 1.92589951
O 0.35195212 4.47324910 0.70294502
O 2.32678304 2.48829365 3.85528783
O 2.32678304 5.02486989 4.03124575
O 2.71943629 5.02486989 1.40240122
O 2.71943629 2.48829365 1.22644331
O 4.69426723 1.93667284 4.55474404
O 4.69426723 4.47324910 3.33178954
```




$HfO_2$ ferroelectric phase structure with polarization pointing upwards, see the *structure.as* file below:

```
Total number of atoms
12
Lattice
5.04621935 0.00000000 0.00000000
0.00000000 5.07315250 0.00000000
0.00000000 0.00000000 5.25768906
Cartesian
Hf 1.34815269 1.31512402 0.17639072
Hf 1.34815269 3.85170026 2.45245381
Hf 3.69806665 1.31512402 2.80523525
Hf 3.69806665 3.85170026 5.08129834
O 0.35195212 0.59990340 1.92589951
O 0.35195212 3.13647965 0.70294502
O 2.32678304 2.58485884 4.03124575
O 2.32678304 5.12143510 3.85528783
O 2.71943630 5.12143510 1.22644331
O 2.71943630 2.58485884 1.40240122
O 4.69426723 0.59990340 4.55474404
O 4.69426723 3.13647965 3.33178954
```




Insert a series of intermediate transition structures between the polarization-down and polarization-up structures using **linear interpolation** (neb.linear_interpolate), as detailed in the utility script *neb_structure.py*. In this example, **11** intermediate structures are inserted, resulting in a total of **13** configurations including the initial and final polarization phases. Polarization calculations are then performed sequentially on all configurations.

### 2.20.2 Run the program.

After preparing the input files, upload *polarization.in* and each *structure.as* file to the server, placing the 13 structures into 13 directories. Then, execute *DS-PAW polarization.in* following the method described in Structure Relaxation.

### 2.20.3 Analysis of calculation results

Based on the input files mentioned above, 13 sets of output files will be generated after the calculation, including *DS-PAW.log*, *scf.h5*, and *polarization.txt*.

- *DS-PAW.log* : The log file generated after the DS-PAW ferroelectric calculation.
- *scf.h5* : The self-consistent field (SCF) calculation output file in **h5** format. Note that the name of the h5 file must strictly match the task type. For h5 file parsing, refer to the *Output File Format Specification* section.
- *polarization.txt* : The **txt** file generated after the ferroelectric polarization calculation is completed. It stores the electronic and ionic contributions to polarization, as well as the total polarization quantum number, for easy access by the user.

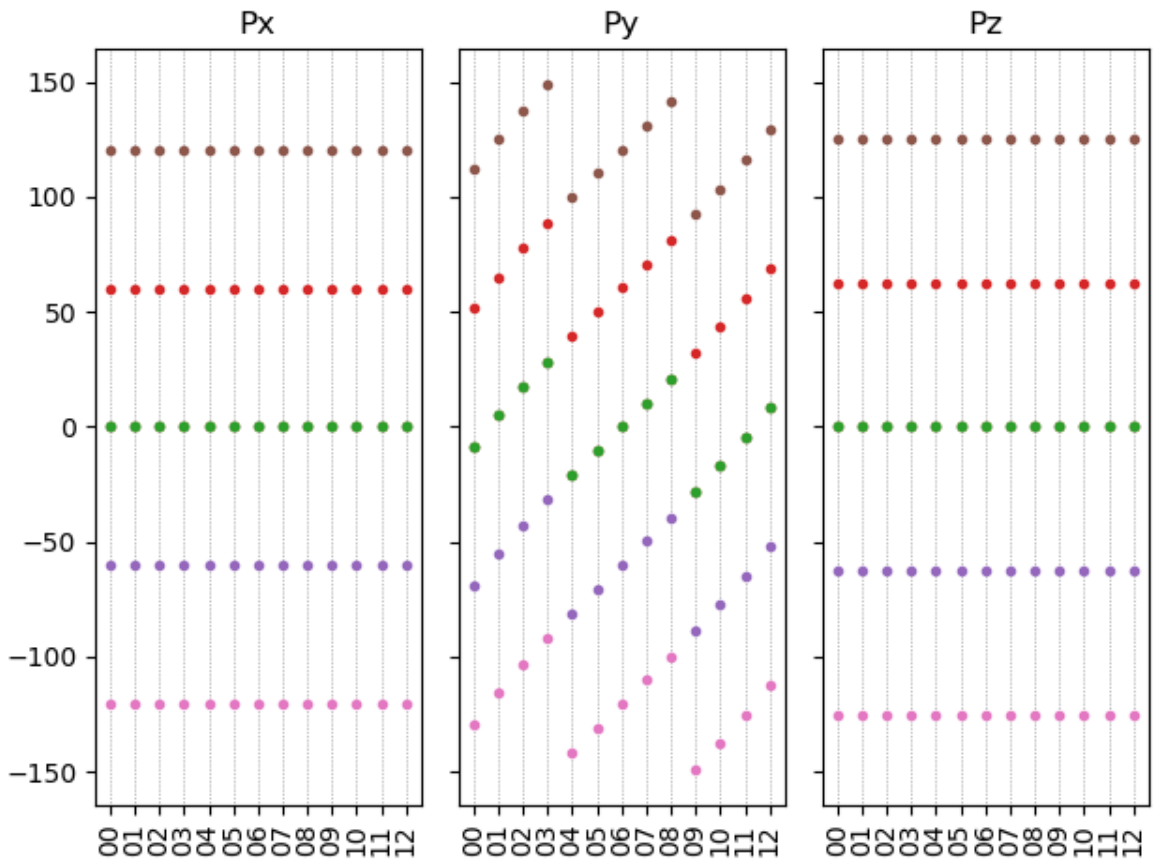
For the ferroelectric phase system with downward polarization (00) as an example, the ferroelectric polarization data of $HfO_2$ can be obtained from the *polarization.txt* file as follows:

| Total(x y z) ( $\mu C/cm^2$ ) | | |
|---|---|---|
| -0.000043 | -8.715604 | -0.000002 |
| Quantum(x y z) ( $\mu C/cm^2$ ) | | |
| 60.067225 | 60.387821 | 62.584436 |

For example, in a polarization-up (12) ferroelectric phase system, the ferroelectric polarization data for $HfO_2$ can be obtained from the *polarization.txt* file as follows:

| Total(x y z) ( $\mu C/cm^2$ ) | | |
|---|---|---|
| -0.000049 | 8.715446 | 0.000001 |
| Quantum(x y z) ( $\mu C/cm^2$ ) | | |
| 60.067225 | 60.387821 | 62.584436 |

The *PolaTotal.py* script can be used to process the *scf.h5* file that writes polarization data. See the *Auxiliary Tool User Guide* section for specific instructions. Processing the data for 13 ferroelectric calculations yields the following result figure:




The figure above shows the polarization intensity Px, Py, and Pz in the x, y, and z directions, obtained after polarization quantum periodic conversion. Since the polarization direction of $HfO_2$ is the y-direction, the values of Px and Pz do not change with atomic displacement.

- Analyzing the group with the polarization number closest to 0 in the Py direction, the polarization intensity value of $HfO_2$ is the difference in polarization number between the ferroelectric phase (downward polarization, sequence number 00 or upward polarization, sequence number 12) and the central symmetric phase (transition state, sequence number 06), combined with the *polarization.txt* file and the polarization data figure above, to obtain:
- The polarization difference between configurations 00 and 06 is $-69.103\mu C/cm^2$
- The polarization difference between configurations 12 and 06 is $69.103\mu C/cm^2$

Therefore, the polarization intensity of $HfO_2$ is $69.103\mu C/cm^2$

## 2.21 Bader Charge Calculation

This section will use the NaCl crystal as an example to introduce how to perform Bader charge calculation in DS-PAW and analyze the valence distribution of each atom in the NaCl system.

### 2.21.1 Input file for Bader charge calculation of $NaCl$ crystal

The input files include the parameter file *bader.in* and the structure file *structure.as*. *bader.in* is as follows:

The *bader.in* file is shown below:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoff = 650
#outputs
io.charge = true
io.wave = false
io.bader = true
```

*bader.in* Input Parameter Description:

This Bader charge calculation is performed based on the self-consistent calculation. In addition to the basic parameters of the self-consistent calculation, the following new parameters are added:

- `io.bader`: Controls the Bader charge calculation switch during the self-consistent calculation;

The *structure.as* file is referenced as follows:

```
Total number of atoms
8
Lattice
5.68452692 0.00000000 0.00000000
0.00000000 5.68452692 0.00000000
0.00000000 0.00000000 5.68452692
Cartesian
Na 4.26339519 1.42113173 1.42113173
Na 1.42113173 4.26339519 1.42113173
Na 1.42113173 1.42113173 4.26339519
Na 4.26339519 4.26339519 4.26339519
Cl 1.42113173 1.42113173 1.42113173
Cl 4.26339519 4.26339519 1.42113173
Cl 4.26339519 1.42113173 4.26339519
Cl 1.42113173 4.26339519 4.26339519
```

> **Note**
>
> When *io.bader* is true, *io.charge* must also be true.

### 2.21.2 Run the program

After preparing the input files, upload the *bader.in* and *structure.as* files to the server for execution. Run *DS-PAW bader.in* following the methods described in the structural relaxation section.

### 2.21.3 Analysis results analysis

Based on the input files mentioned above, after the calculation is complete, output files such as *DS-PAW.log*, *scf.h5*, and *bader.txt* will be generated.

- *DS-PAW.log* : The log file generated after the DS-PAW Bader charge calculation.
- *scf.h5* : The self-consistent field (SCF) calculation output file in **h5** format. Note that the name of the h5 file must strictly match the task type. For h5 file parsing, see the specific data structure details in the *Output File Format Specification* section;
- *bader.txt* : The **txt** file generated after the Bader charge calculation, containing Bader charge data for quick access by users.

The content of the *bader.txt* file is as follows, and the data obtained from the Bader charge analysis is consistent with the data from the Henkelman group at the University of Texas at Austin.

Total number of valence electronics: 64

| Element | X | Y | Z | Charge | AtomicVolume | MinDistance |
|---|---|---|---|---|---|---|
| Cl | 0.25 | 0.25 | 0.25 | 7.85852 | 35.893 | 1.65799 |
| Cl | 0.75 | 0.75 | 0.25 | 7.85704 | 35.83 | 1.65799 |
| Cl | 0.75 | 0.25 | 0.75 | 7.84024 | 35.0495 | 1.65799 |
| Cl | 0.25 | 0.75 | 0.75 | 7.87537 | 36.6765 | 1.65799 |
| Na | 0.75 | 0.25 | 0.25 | 8.14221 | 10.0598 | 1.10532 |
| Na | 0.25 | 0.75 | 0.25 | 8.14223 | 10.0607 | 1.10532 |
| Na | 0.25 | 0.25 | 0.75 | 8.14221 | 10.0598 | 1.10532 |
| Na | 0.75 | 0.75 | 0.75 | 8.14221 | 10.0598 | 1.10532 |

## 2.22 bandunfolding calculation

This section will use the $Cu_3Au$ system as an example to demonstrate how to perform band unfolding calculations in DS-PAW, and analyze the band structure of $Cu_3Au$ after unfolding.

### 2.22.1 $Cu_3Au$ band unfolding calculation input file

Band unfolding calculations require a two-step band calculation. Therefore, the input files include the parameter files *scf.in*, *bandunfolding.in*, and the structure file *structure.as*.

The *scf.in* file is as follows:

```
task = scf

sys.structure = structure.as
sys.symmetry = true
```

(continues on next page)

(continued from previous page)

```
sys.functional = PBE
sys.spin = none

cal.methods = 1
cal.smearing = 1
cal.ksamping = MP
cal.kpoints = [3, 3, 3]
cal.cutoff = 650

scf.convergence = 1.0e-05

io.charge = true
io.wave = false
```

*bandunfolding.in* as follows:

```
task = band
cal.iniCharge = ./rho.bin

sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

cal.methods = 1
cal.smearing = 1
cal.ksamping = MP
cal.kpoints = [3, 3, 3]
cal.cutoff = 500

scf.convergence = 1.0e-05

band.unfolding = true
band.primitiveUVW=[0.0, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.0]
band.kpointsLabel= [R,G,X]
band.kpointsCoord= [0.5, 0.5, 0.5, 0.0, 0.0, 0.0, 0.5, 0.0, 0.5]
band.kpointsNumber= [101, 101]

io.charge = false
io.wave = false
```

*bandunfolding.in* Input parameter introduction:

The band unfolding calculation is performed based on the band calculation, and the band calculation must be completed in a two-step process. In addition to the basic parameters for band calculation, the new parameters are as follows:

- `band.unfolding` : Controls the switch for band unfolding calculation in band structure calculations;
- `band.primitiveUVW` : Sets the UVW coefficients. Multiplying the supercell lattice vectors by the UVW coefficients results in the primitive cell lattice vectors, which is used to control the band unfolding parameters.

The *structure.as* file is referenced as follows:

```
Total number of atoms
4
Lattice
3.7530000210         0.0000000000         0.0000000000
0.0000000000         3.7530000210         0.0000000000
0.0000000000         0.0000000000         3.7530000210
Direct
Au     0.000000000         0.000000000         0.000000000
Cu     0.000000000         0.500000000         0.500000000
Cu     0.500000000         0.000000000         0.500000000
Cu     0.500000000         0.500000000         0.000000000
```

### 2.22.2 run program execution

After preparing the input files, upload the *scf.in*, *bandunfolding.in*, and *structure.as* files to the server and run them. Execute *DS-PAW scf.in* as described in the structure relaxation section. After the self-consistent calculation is completed, execute *DS-PAW bandunfolding.in*.

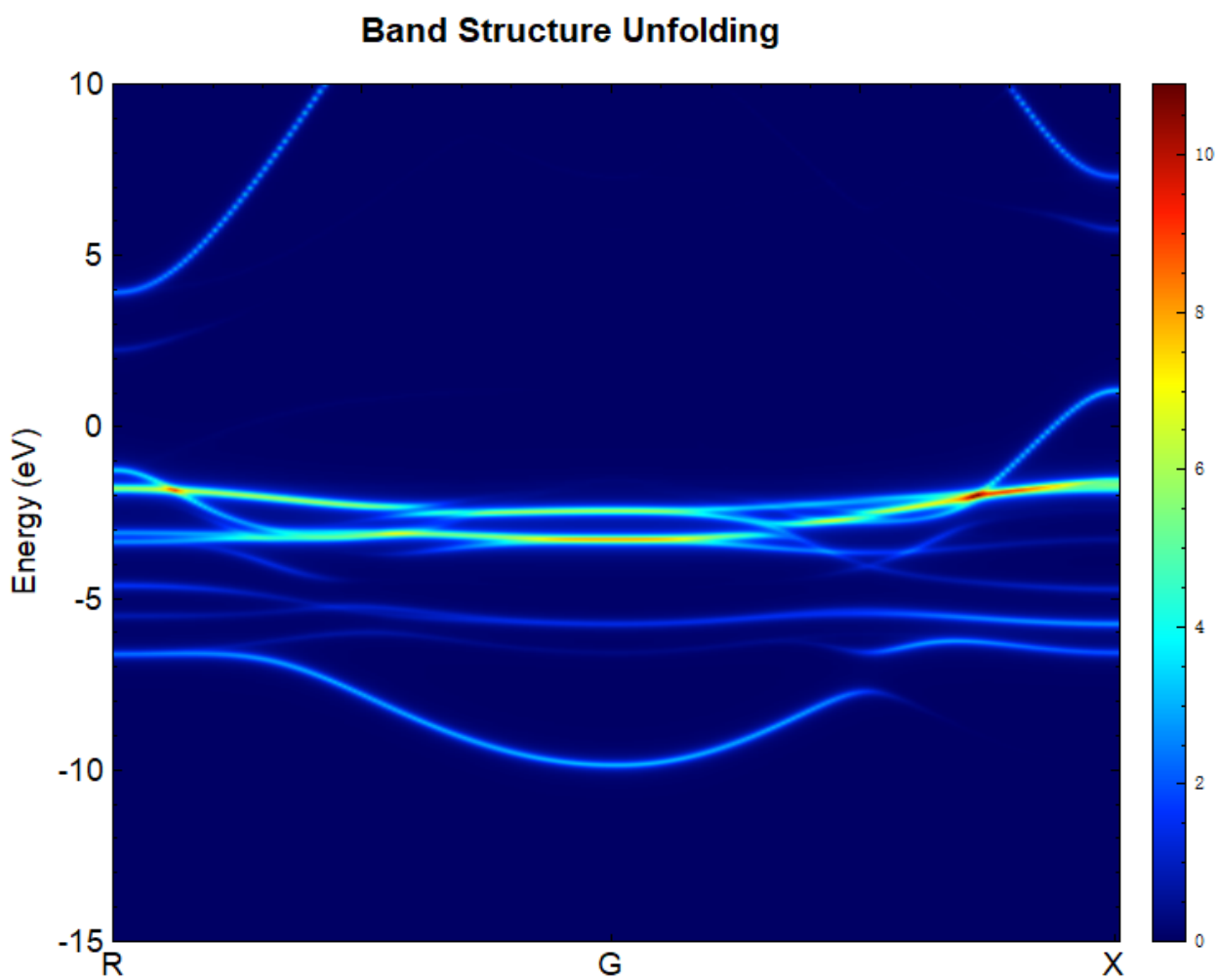
### 2.22.3 Analysis of calculation results.

Based on the input files mentioned above, the calculation will generate output files such as *DS-PAW.log*, *scf.h5*, and *band.h5*.

*:guilabel:`band.h5* : The **h5** output file corresponding to the band structure calculation. Compared to the band structure calculation, this file adds the **UnfoldingBandInfo** section. See *Output File Format Specification* for a detailed structure analysis.

The script *bandunfolding.py* can be used to process data from *band.h5*. See *Auxiliary Tool User Guide* for detailed instructions. The resulting band structure plot should look like the following figure, consistent with the results reported in[1].

[1] Mingxing Chen and M.ăWeinert. Layer k-projection and unfolding electronic bands at interfaces. *Phys. Rev. B*, 98:245421, Dec 2018. doi:10.1103/PhysRevB.98.245421.




## 2.23 epsilon Dielectric Constant Calculation

This section will use the Si system as an example to introduce how to perform dielectric constant calculations in DS-PAW.

### 2.23.1 $Si$ dielectric constant calculation input file

The input files consist of the parameter file *epsilon.in* and the structure file *structure.as*. The contents of *epsilon.in* are as follows:

```
# task type
task = epsilon
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
```

(continues on next page)

(continued from previous page)

```
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [5, 5, 5]
cal.cutoff = 500
scf.convergence = 1.0e-7
```

*epsilon.in* Input parameter introduction:

The calculation of dielectric constants can be performed by directly specifying **task**. The new optional values for task are as follows:

- `task` : Sets the calculation type. Adds the **epsilon** parameter, which corresponds to the calculation of the dielectric constant here.

> **Note**
>
> Dielectric constant calculations are also possible when `task = phonon` and `phonon.method = dfpt` by adding the parameter `phonon.dfptEpsilon = true`.

The *structure.as* file is referenced as follows:

```
Total number of atoms
8
Lattice
   5.43070000 0.00000000 0.00000000
   0.00000000 5.43070000 0.00000000
   0.00000000 0.00000000 5.43070000
Cartesian
Si 0.67883750 0.67883750 0.67883750
Si 3.39418750 3.39418750 0.67883750
Si 3.39418750 0.67883750 3.39418750
Si 0.67883750 3.39418750 3.39418750
Si 2.03651250 2.03651250 2.03651250
Si 4.75186250 4.75186250 2.03651250
Si 4.75186250 2.03651250 4.75186250
Si 2.03651250 4.75186250 4.75186250
```

### 2.23.2 run program execution

Once the input files are ready, upload the *epsilon.in* and *structure.as* files to the server and run the *DS-PAW epsilon.in* as described in the structure relaxation section.

### 2.23.3 Analysis of calculation results

Based on the input files mentioned above, the calculation will generate output files such as *DS-PAW.log*, *epsilon.h5*, and *epsilon.txt*.

- *DS-PAW.log* : The log file generated after the DS-PAW dielectric constant calculation;
- *epsilon.h5* : The **h5** output file corresponding to the dielectric constant calculation. For the specific data structure, see the *Output File Format Specification* section;
- *epsilon.txt* : The **txt** text file generated after the dielectric constant calculation, which writes data related to the dielectric constant for quick user access.

The following data can be obtained from the *epsilon.txt* file:

| Total Part | | |
|---|---|---|
| 13.309902 | 0.000000 | -0.000000 |
| -0.000000 | 13.309902 | -0.000000 |
| -0.000000 | 0.000000 | 13.309902 |

Analyzing the table above, the dielectric constant of the system is **13.309902**, which is consistent with the literature value of **13.31** reported in :footcite:p:*PhysRevB.73.045112*.

## 2.24 Piezoelectric Tensor Calculation

This section will demonstrate how to calculate the piezoelectric tensor, specifically obtaining the piezoelectric coefficient $e_{33}(0)$ for a material, using AlN as an example within the DS-PAW framework.

### 2.24.1 Input file for piezoelectric tensor calculation of $AlN$

The input file contains the parameter file *piezo.in* and the structure file *structure.as*, with *piezo.in* as follows:

```
task = epsilon
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [10, 10, 10]
cal.cutoffFactor = 1.5
scf.convergence = 1.0e-7

#outputs
io.charge = false
io.wave = false
```

*piezo.in* Input parameter introduction:

- `task` : Sets the calculation type, adding the **epsilon** parameter; here, it corresponds to the piezoelectric tensor calculation.
- `scf.convergence` : Sets the precision for electronic convergence in dielectric tensor calculations; it is recommended to increase the precision, which is set to 1.0e-7 here.

*structure.as* file is referenced as follows:

```
Total number of atoms
8
Lattice
3.11606630 0.00000000 0.00000000
0.00000000 5.39683518 0.00000000
0.00000000 0.00000000 5.00770902
```

(continues on next page)

(continued from previous page)

```
Cartesian
Al 0.00000000 3.59735137 0.00946380
Al 0.00000000 1.79945276 2.51320124
Al 1.55803315 0.89899597 0.00945662
Al 1.55803315 4.49786165 2.51308138
N 0.00000000 3.59851112 1.91845914
N 0.00000000 1.79831356 4.42266820
N 1.55803315 0.90013952 1.91851680
N 1.55803315 4.49672497 4.42258192
```

### 2.24.2 run program execution

Once the input files are ready, upload the files *piezo.in* and *structure.as* to the server and run *DS-PAW piezo.in* following the method described in Structure Relaxation.

### 2.24.3 Analysis of calculation results

Based on the input files mentioned above, the calculation will produce the following output files: *DS-PAW.log*, *epsilon.h5*, and *epsilon.txt*.

- *DS-PAW.log* : The log file generated after the DS-PAW piezoelectric tensor calculation.
- *epsilon.h5* : The **h5** output file corresponding to the dielectric constant calculation, and the specific data structure can be found in the *Output File Format Specification* section;
- *epsilon.txt* : The **txt** text file after the piezoelectric calculation is completed. This file writes piezoelectric-related data for users to quickly obtain information.

The following data can be obtained from the *epsilon.txt* file:

| Piezoelectric Tensor (C/m^2)( Row: x y z Column: XX YY ZZ XY YZ ZX) | | | | | |
|---|---|---|---|---|---|
| Electronic Part | | | | | |
| 0.000000 | 0.000000 | 0.000000 | 0.000006 | 0.000000 | 0.336610 |
| -0.000001 | 0.000007 | 0.000003 | 0.000000 | 0.336662 | 0.000000 |
| 0.266339 | 0.265888 | -0.419569 | 0.000000 | -0.000014 | 0.000000 |
| Ionic Part: | | | | | |
| -0.000004 | 0.000002 | 0.000002 | 0.000032 | -0.000000 | -0.681702 |
| -0.000163 | -0.000239 | 0.000314 | -0.000000 | -0.699012 | -0.000000 |
| -0.911456 | -0.913265 | 1.943887 | -0.000000 | -0.000633 | -0.000000 |
| Total Part: | | | | | |
| -0.000004 | 0.000002 | 0.000002 | 0.000039 | -0.000000 | -0.345092 |
| -0.000164 | -0.000232 | 0.000317 | -0.000000 | -0.362350 | -0.000000 |
| -0.645117 | -0.647377 | 1.524318 | -0.000000 | -0.000647 | -0.000000 |

Analyzing the above table, the value of the piezoelectric tensor electronic contribution $e_{33}(0)$ is **-0.419569** $C/m^2$, and the total piezoelectric tensor $e_{33}$ is **1.524318** $C/m^2$, which is close to the literature values[2] of **-0.47** $C/m^2$ and **1.46** $C/m^2$.

[2] Fabio Bernardini, Vincenzo Fiorentini, and David Vanderbilt. Spontaneous polarization and piezoelectric constants of iii-v nitrides. *Phys. Rev. B*, 56:R10024–R10027, Oct 1997. doi:10.1103/PhysRevB.56.R10024.

## 2.25 fixcell Fixed Basis Vector Relaxation Calculation

This section will use the $MoS_2$ system as an example to introduce how to perform fixed-lattice relaxation calculations in DS-PAW.

### 2.25.1 Fixed-basis-vector relaxation calculation input file for $MoS_2$

The input file consists of the parameter file *relax.in* and the structure file *structure.as*, with *relax.in* as follows:

```
# task type
task = relax
#system related
sys.structure = structure.as
sys.symmetry = false
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.cutoff = 650
cal.kpoints = [19, 19, 5]
#relax related
relax.freedom = all
relax.convergence =  0.05
relax.methods = CG
```

The *structure.as* file is referenced as follows:

```
Total number of atoms
6
Lattice  Fix_x Fix_y Fix_z
3.19031572 0.00000000 0.00000000   F T T
-1.59515786 2.76289446 0.00000000  F F T
0.00000000 0.00000000 14.87900448  T T T
Cartesian
S 0.00000000 1.84193052 12.72413785
S 1.59515943 0.92096386 5.28463561
S 0.00000000 1.84193052 9.59436887
S 1.59515943 0.92096386 2.15486663
Mo 1.59515943 0.92096386 11.15925336
Mo 0.00000000 1.84193052 3.71975112
```

Introduction to the *structure.as* tag settings:

To perform relaxation calculations with fixed cell dimensions, you need to add the fix tags in the *structure.as* file, similar to the fix tag settings for atomic relaxation (adding the Fix tag after the atomic coordinates). To fix the lattice vectors, add the Fix tag after the **Lattice** line on the third line of the *structure.as* file. In this case, the tags correspond to fixing the c-axis and the y and z directions of the a-axis, and the z direction of the b-axis of the cell.

### 2.25.2 Run the program.

After preparing the input files, upload the *relax.in* and *structure.as* files to the server and run them, executing *DS-PAW relax.in* as described in the structure relaxation section.

### 2.25.3 Analysis of calculation results

Based on the input files mentioned above, after the calculation is completed, output files such as *DS-PAW.log*, *relax.h5*, and *latestStructure.as* will be generated.

- *relax.h5*: The corresponding **h5** output file for the relaxation calculation;
- *latestStructure.as*: The final structure file in *.as* format after relaxation, allowing direct data viewing;

Drag *latestStructure.as* into Device Studio to view the structure, or open the file directly to see the structural data after relaxation ends, as follows:

```
Total number of atoms
6
Lattice
3.19696732   0.00000000   0.00000000
-1.59848077   2.76865753   0.00000000
0.00000000   0.00000000  14.87900448
Direct
Mo   0.66666701   0.33333316   0.74999995
Mo   0.33333340   0.66666675   0.24999997
S   0.33333340   0.66666666   0.85535854
S   0.66666686   0.33333303   0.35535875
S   0.33333367   0.66666699   0.64464148
S   0.66666708   0.33333333   0.14464130
```

Comparing the results, before relaxation a = b = **3.19031572**, after relaxation a = b = **3.19696732**, while c = **14.87900448** remained unchanged.

## 2.26 Calculation of Thermodynamic Properties of Phonons for Thermal Transport

This section will use a Si system as an example to introduce how to perform phonon thermodynamic property calculations in DS-PAW.

### 2.26.1 Input file for phonon thermodynamic properties calculation of $Si$

The input files include a parameter file, *phonon-thermal.in*, and a structure file, *structure.as*. *phonon-thermal.in* is shown below:

```
# task type
task = phonon
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
cal.methods = 1
```

(continues on next page)

(continued from previous page)

```
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [5, 5, 5]
cal.cutoffFactor = 1.5
scf.convergence = 1.0e-7
#phonon related
phonon.structureSize = [2,2,2]
phonon.type =dos
phonon.qpoints = [31,31,31]
phonon.method = dfpt

phonon.thermal=true
phonon.thermalRange = [0,1000,10]
```

Input parameters description for *phonon-thermal.in*:

- `phonon.thermal`: Controls the thermodynamic calculation switch in phonon calculations; effective only when *phonon.method = dfpt*.
- `phonon.thermalRange` : Sets the temperature range and data storage interval for thermodynamic calculations;

The following describes the *structure.as* file:

```
Total number of atoms
2
Lattice
0.00 2.75 2.75
2.75 0.00 2.75
2.75 2.75 0.00
Direct
Si -0.125000000 -0.125000000 -0.125000000
Si 0.125000000 0.125000000 0.125000000
```

### 2.26.2 Run the program

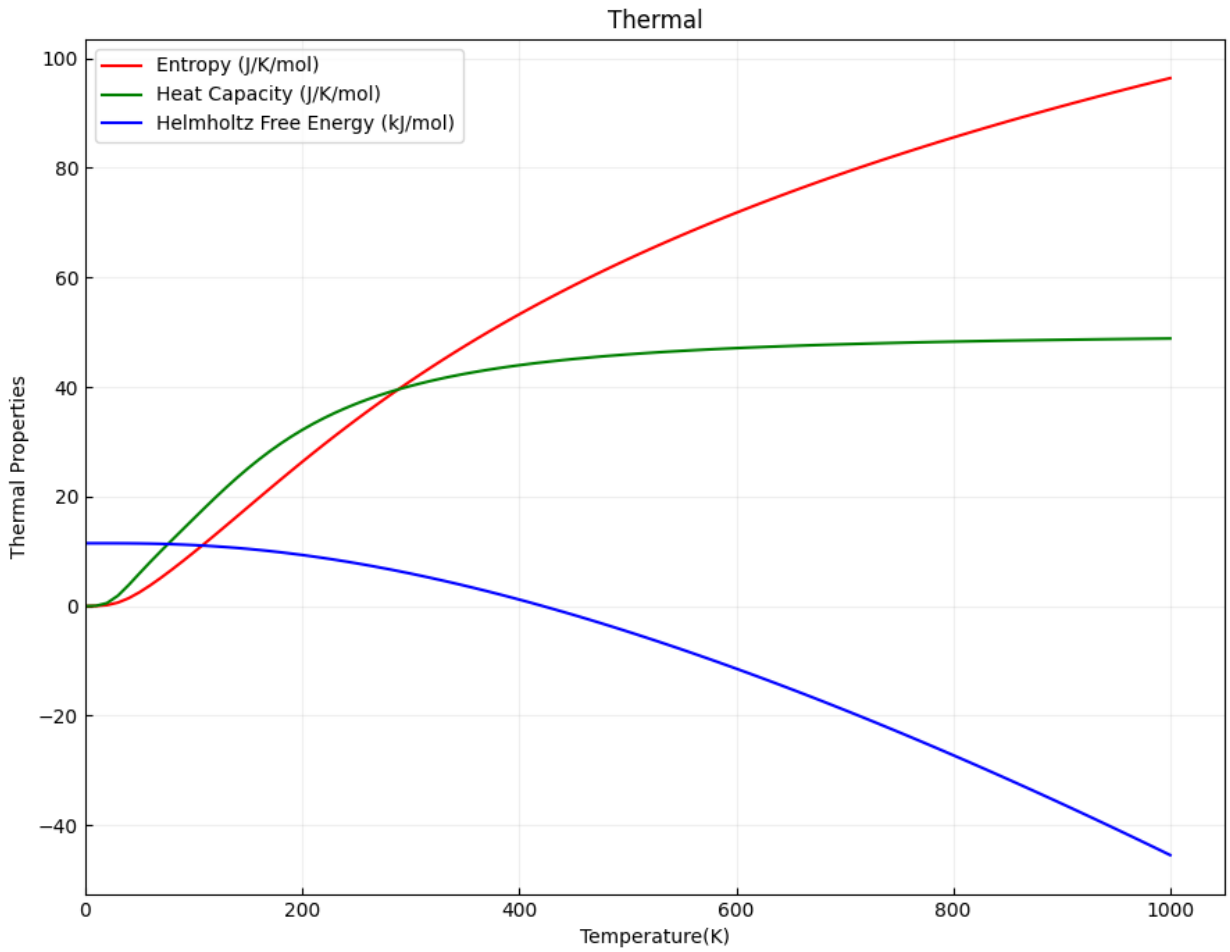
After preparing the input files, upload the *phonon-thermal.in* and *structure.as* files to the server for execution. Run *DS-PAW phonon-thermal.in* following the method described in the structural relaxation section.

### 2.26.3 Results Analysis

Based on the input files mentioned above, after the calculation is complete, output files such as *DS-PAW.log* and *phonon.h5* will be generated.

- *DS-PAW.log*: The log file generated from the DS-PAW phonon calculation.
- *phonon.h5*: The **h5** output file from the DS-PAW phonon calculation. Enabling thermodynamic calculations will write **ThermalInfo** data to the generated *phonon.h5* file. See *Output File Format Specification* for details.

The phonon thermodynamic data can be processed using the *phonon_thermal.py* script, as detailed in the *Auxiliary Tool User Guide* section. Analyzing the thermodynamic data yields curves of entropy, heat capacity, and Helmholtz free energy as a function of temperature, which are consistent with the results presented in the phonopy git repository:




## 2.27 solid state NEB calculation

This section will use the HfZrO system as an example to illustrate how to perform solid state NEB calculations with cell relaxation within DS-PAW.

### 2.27.1 $HfZrO$ Solid state NEB input file

The input files include the parameter file *ssneb.in* and the structure file *structure.as*, with the contents of *ssneb.in* as follows:

```
task = neb

sys.structure = structure.as
sys.functional  = LDA
sys.spin = none
sys.symmetry = false

cal.ksamping = G
cal.kpoints = [10,10,10]
cal.cutoff = 650
cal.methods = 1
cal.smearing = 1
```

(continues on next page)

(continued from previous page)

```
cal.sigma = 0.05

scf.mixType = Broyden
scf.mixBeta = 0.4
scf.convergence = 1e-6
scf.max = 300

neb.springK = 5
neb.images = 6
neb.iniFin = true
neb.method = QM2
neb.convergence = 0.01
neb.max = 500
neb.freedom = all

io.wave = false
io.charge = false
```

*ssneb.in* Input Parameter Introduction:

- `neb.freedom` : Specifies the dimensions for transition state relaxation. Setting it to all corresponds to relaxing the cell size;
- `neb.method` : Sets the method for transition state search. When `neb.freedom` = all, the available options for this parameter are QM2 and FIRE;

*structure.as* must be provided multiple times, and the initial state structure *structure00.as* is referenced as follows:

```
Total number of atoms
12
Lattice
5.00209138 0.00000009 0.00000004
0.00000009 5.00209143 -0.00000004
0.00000004 -0.00000004 5.07896990
Cartesian
Hf 2.50104558 2.50104575 0.00000000
Hf 0.00000000 0.00000000 0.00000000
O 3.75156841 1.25052303 1.47285183
O 3.75156857 3.75156869 1.04735062
O 1.25052293 1.25052297 3.60611823
O 1.25052286 3.75156867 4.03161932
O 1.25052287 3.75156860 1.47285187
O 1.25052275 1.25052294 1.04735054
O 3.75156850 1.25052287 4.03161945
O 3.75156850 3.75156869 3.60611821
Zr 2.50104577 0.00000000 2.53948497
Zr 0.00000000 2.50104594 2.53948491
```

Final state structure: See *structure07.as* below.

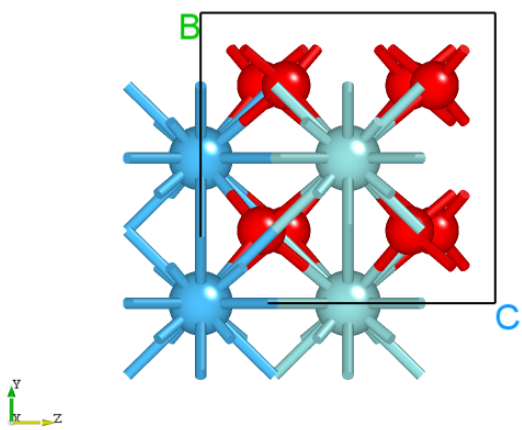
```
Total number of atoms
12
Lattice
4.98221520 -0.00002552 0.00036684
```

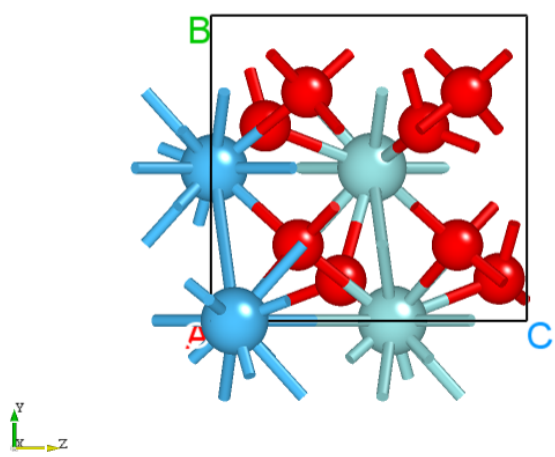
(continues on next page)

(continued from previous page)

```
-0.00002562 4.99587652 0.00005905
0.00039053 0.00006126 5.18258321
Cartesian
Hf 2.30823006 2.49975412 0.04967381
Hf 0.00919001 0.00195723 0.38722458
O 4.03365086 0.66419181 2.12958714
O 4.00001549 3.18954023 0.89210846
O 0.95871628 1.24120307 4.04442128
O 0.94984693 3.74053908 4.19050825
O 1.35895285 3.73907584 1.57483409
O 1.36804279 1.24264997 1.42944278
O 3.29999107 0.69159253 4.72728663
O 3.26626721 3.16200890 3.48972595
Zr 2.31915914 0.00841995 2.97686955
Zr 4.98082249 2.50639160 2.64290889
```

The initial and final state configurations are displayed in **Device Studio** as follows:



**Note**

1. When `neb.freedom = all`, the options for `neb.method` are **QM2** or **FIRE**.
2. The generation of intermediate structures can be done by calling the `neb_interpolate_structures.py` script, as described in the Auxiliary Tools Tutorial - Transition State Section. After interpolation, the `neb_visualize.py` script can be called to preview the interpolated structures, and the `calc_dist.py` script can be used to check if the distances between images are reasonable.

### 2.27.2 Run the program.

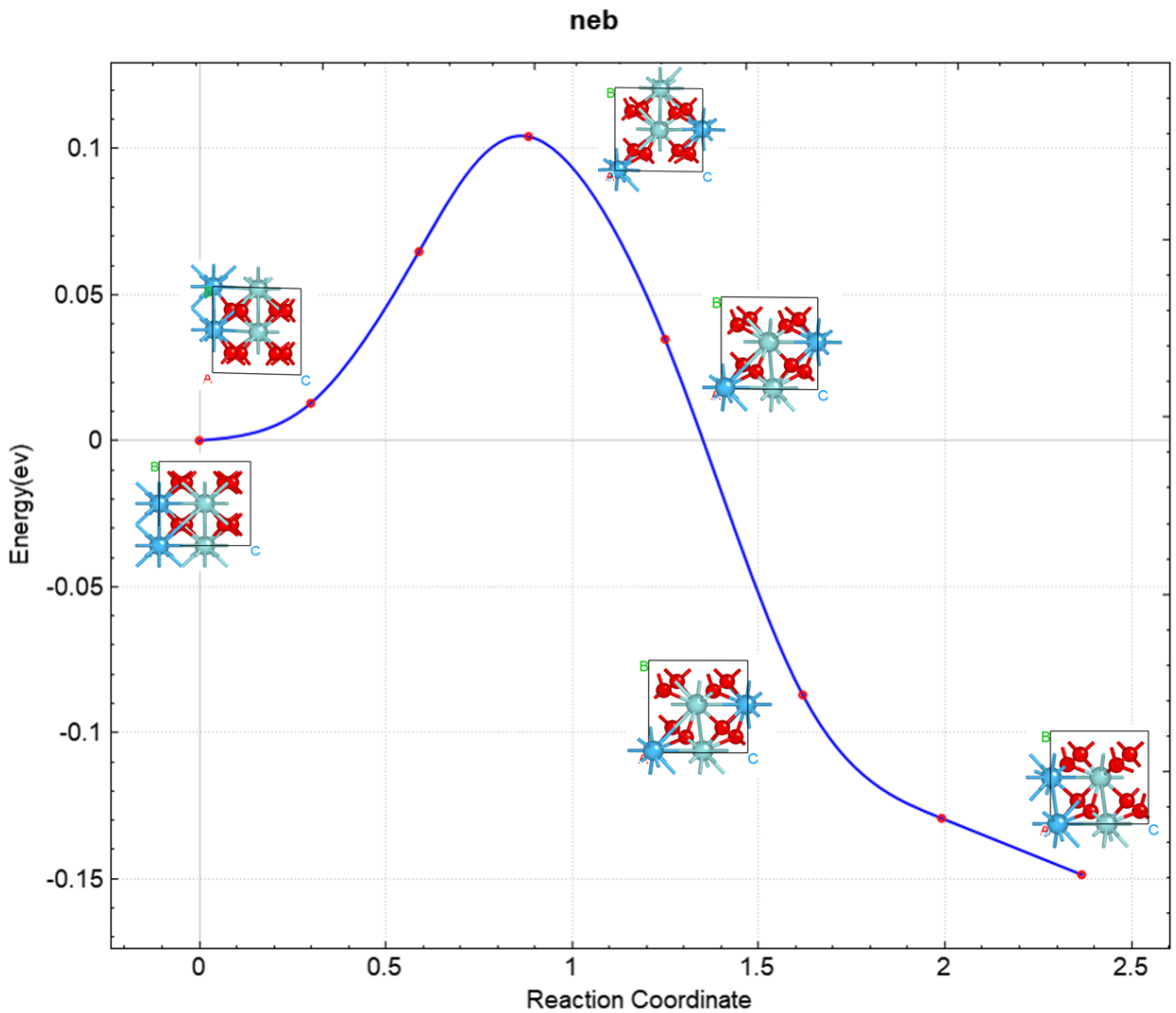
Once the input files are prepared, upload the *ssneb.in* and the folders containing *structureNo.as* files to the server and run the *DS-PAW ssneb.in* command as described in the structure relaxation section.

### 2.27.3 Analysis of the calculation results

After the calculation is completed based on the input files described above:

- The folders containing the initial and final state structures will generate the self-consistent calculation output files such as *DS-PAW.log*, *latestStructure00.as*, and *scf.h5*;
- The folders `No` containing the intermediate structures *structureNo.as* (folders containing intermediate structures involved in the transition state calculation, the number of intermediate structures is determined by the `neb.images` parameter) will generate output files such as *nebNo.h5* and *latestStructureNo.as* from the structure optimization.
- The outermost directory will generate the files *DS-PAW.log* and *neb.h5*. The file *neb.h5* is a summary of the information in the *nebNo.h5* files located in the `No` folders.
- *DS-PAW.log* : Log file generated after DS-PAW transition state calculation.
- *neb.h5* : The **h5** data file after the transition state calculation is completed; the reaction coordinate, energy changes, and other data are saved in *neb.h5*. For details on the data structure, see the *Output File Format Specification* section.

The results of the NEB calculation can be analyzed using the **python** script *neb.py*. The analysis script should be executed within the complete NEB calculation directory. See the *Auxiliary Tool User Guide* section for specific instructions. The resulting reaction barrier curve should look like this:




## 2.28 Solvation Energy Calculation

This section will use the $H_2O$ system as an example to demonstrate how to calculate solvation energy under the implicit solvent model in DS-PAW.

### 2.28.1 $H_2O$ solvation energy calculation input file

The input file includes the parameter file *scf.in* and the structure file *structure.as*, with *scf.in* as follows:

```
# task type
task = scf
#system related
sys.structure = structure.as
sys.symmetry = true
sys.functional = PBE
sys.spin = none

#scf related
```

(continues on next page)

(continued from previous page)

```
cal.methods = 1
cal.smearing = 3
cal.sigma = 0.2
cal.ksamping = G
cal.kpoints = [1, 1, 1]
cal.supGrid = true
cal.cutoff = 800
scf.convergence = 1.0e-6

#implicit solvation model
sys.sol = true
sys.solEpsilon = 80
sys.solTAU = 5.25E-4

#outputs
io.charge = false
io.wave = false
io.boundCharge = true
```

*scf.in* input parameters:

- `sys.sol`: Controls the switch for introducing the implicit solvation model. If true, solvation effects are considered.
- `sys.solEpsilon`: Sets the magnitude of the solvent dielectric constant, set to 80 in this example;
- `sys.solTAU` : Specifies the magnitude of the effective interfacial tension per unit area, in units of eV/Å^2, with a default value of 5.25E-4. It is recommended to set this parameter to a value less than 1e-3.
- `io.boundCharge`: Switch for controlling the output of solvent-bound charge density files.

Structure file: referenced as *structure.as* as follows

```
Total number of atoms
3
Lattice
10.00000000 0.00000000 0.00000000
0.00000000 10.00000000 0.00000000
0.00000000 0.00000000 10.00000000
Cartesian
H 5.63934499 4.89541998 4.58224001
H 4.36065501 5.11499002 5.45934003
O 4.65002501 4.88500998 4.54065997
```

### 2.28.2 run program execution

Once the input files are prepared, upload the *:guilabel:`scf.in* and *:guilabel:`structure.as* files to the server and run them, executing *:guilabel:`DS-PAW scf.in* according to the method described in Structure Relaxation.

### 2.28.3 Analysis of the calculation results

Based on the input files, the calculation will generate output files including *DS-PAW.log*, *scf.h5*, and *rhoBound.h5*.

- *DS-PAW.log* : The log file generated after the DS-PAW implicit solvation model calculation.
- *scf.h5* : The self-consistent field (SCF) calculation output file in **h5** format;

- *rhoBound.h5* : The solvent-bound charge density file obtained from implicit solvent calculations.

This example calculates the total energy $E_{sol}$ including the solvation energy, which is calculated as follows:

Solvation Energy=E(sys.sol=true)-E(sys.sol=false)

Based on this formula, a separate calculation with sys.sol = false is required to obtain the total energy without solvation, $E_{nosol}$. Substituting $E_{nosol}$ into the above formula yields a solvation energy of **-0.313** eV for water, which is consistent with the literature value reported in[3].

When performing calculations with the implicit solvation model, the solvent-bound charge density distribution file around the solute, *rhoBound.h5*, can also be obtained. This file can be post-processed using the **python** script *trans_rho.py*. For specific operations, refer to the *Auxiliary Tool User Guide* section. The converted visualization file can be opened in VESTA, resulting in the following isodensity surface distribution map:


As can be seen from the figure, the distribution of solvated positive and negative shielded charge densities is located around the water molecules, forming a solvation shell, which is consistent with the models expectations and similar to the distribution of solvent-bound charge densities calculated by other software.

## 2.29 fixedpotential fixed potential calculation

This section will demonstrate how to perform fixed potential calculations in DS-PAW, using the $Cu-slab$ system as an example.

### 2.29.1 $Cu-slab$ Fixed Potential Calculation Input File

The input files include the parameter file *fixedP.in* and the structure file *structure.as*. The content of *fixedP.in* is as follows:

```
# task type
task = scf

sys.functional = PBE
```

(continues on next page)

[3] Kiran Mathew, Ravishankar Sundararaman, Kendra Letchworth-Weaver, TAăArias, and RichardăG Hennig. Implicit solvation model for density-functional study of nanocrystal surfaces and reaction pathways. *The Journal of chemical physics*, 140(8):084106, 2014. doi:10.1063/1.4865107.

(continued from previous page)

```
sys.structure = structure.as

cal.ksamping = G
cal.cutoff = 650
cal.sigma = 0.2
cal.smearing = 3
cal.kpoints = [7,7,1]

scf.convergence = 1.0e-6
scf.max = 200

sys.sol = true
sys.solEpsilon = 78.4
sys.solLambdaD = 3.04
sys.solTAU = 0

# Potential fixed
sys.fixedP = true
sys.fixedPPotential = 2.155

io.charge = true
io.wave = false
```

Introduction to input parameters for `fixedP.in`:

- `task`: Sets the calculation type; in this example, a fixed potential calculation is performed when task=scf.
- `sys.sol`: Enables the solvation model. Fixed potential calculations need to be performed based on the implicit solvation model.
- `sys.solEpsilon` : Sets the solvent dielectric constant, which is set to 78.4 in this example.
- `sys.solLambdaD` : Uses the Poisson-Boltzmann equation and sets the Debye length. If not set, the Poisson equation is used, which does not account for the contribution of interfacial ions to the electrostatic potential.
- `sys.solTAU` : Specifies the effective interfacial tension per unit area, in units of eV/Å^2. The default value is 5.25E-4, and it is recommended to set this parameter to a value less than 1e-3.
- `sys.fixedP` : Enables the fixed potential calculation;
- `sys.fixedPPotential` : Sets the potential value for the fixed potential calculation, defaulting to the Standard Hydrogen Electrode (SHE) as the reference electrode potential. To use the Potential of Zero Charge (PZC) as the reference electrode, set the parameter `sys.fixedPType = PZC`;

**Note**

1. Regarding the Debye length `sys.solLambdaD`, its expression is $\lambda_D = \sqrt{\frac{\varepsilon\varepsilon_o k_B T}{2c^0 z^2 q^2}}$

   The Debye length for a 1M aqueous solution of +/-1 charged ions is: 3.04 Å

The *structure.as* file is referenced as follows:

```
Total number of atoms
8
```

(continues on next page)

(continued from previous page)

```
Lattice
3.63404989 0.00000000 0.00000000
0.00000000 3.63404989 0.00000000
0.00000000 0.00000000 23.62132454
Cartesian
Cu 0.00000000 0.00000000 1.81702310
Cu 1.81702495 0.00000000 3.63404620
Cu 1.81702495 1.81702495 1.81702310
Cu 0.00000000 1.81702495 3.63404620
Cu 0.00000000 0.00000000 5.46390548
Cu 1.81702495 0.00000000 7.22885308
Cu 1.81702495 1.81702495 5.46390548
Cu 0.00000000 1.81702495 7.22885308
```

### 2.29.2 Run the program.

After preparing the input files, upload the *:guilabel: `fixedP.in` ` and *:guilabel: `structure.as` ` files to the server and run them, executing *:guilabel: `DS-PAW fixedP.in` ` as described in Structure Relaxation.

### 2.29.3 Analysis of calculation results

Based on the input files described above, the calculation will generate output files such as *DS-PAW.log* and *scf.h5*.

- *DS-PAW.log* : The log file generated after the DS-PAW calculation with a fixed potential is completed.
- *scf.h5* : The **h5** output file corresponding to DS-PAW when task equals scf; for the specific data structure, please refer to section *Output File Format Specification*.

DS-PAW employs the steepest descent method for fixed-potential calculations, iteratively calculating the charge of the system for self-consistent field (SCF) calculations. The convergence process of multiple SCF calculations is written to the *DS-PAW.log* file. In this example, the convergence criterion is met at LOOP 5. The potential values of the system at the end of LOOP 5 are shown below:

```
## FINISHED FIXEDPOTENTIAL LOOP 5 ##
Electron                      : 149.993000
ElectrodePotential_SHE        : 2.157747 V
ElectrodePotential_PZC        : 2.484286 V
ElectrodePotential_SHE(PZC)   : -0.326539 V
Chemical Potential(electron)  : -6.757747 eV
Grand Total Energy(sigma->0)  : -43088.518081 eV
```

Where

**Electron** is the number of electrons in the system at the end of the iteration;

**ElectrodePotential_SHE** is the electrode potential of the system at the end of the iteration relative to the Standard Hydrogen Electrode (SHE);

**ElectrodePotential_PZC** is the potential of the system at the end of the iteration relative to the point of zero charge (PZC);

**ElectrodePotential_SHE(PZC)** provides the electrode potential of the system at the point of zero charge (PZC) relative to the Standard Hydrogen Electrode (SHE);

**Chemical Potential(electron)** provides the chemical potential of electrons at the iteration endpoint (with the potential at the center of the implicitly solvated solution set to zero).

**Grand Total Energy(sigma->0)** Gives the total energy of the system at the iteration endpoint under the grand canonical ensemble, which is related to the total energy of the system, the change in the number of electrons, and the electron chemical potential.

The calculated potential of the system at the end of the calculation is **2.157** V, which is close to the target potential of **2.155** V.

> **Note**
>
> 1. **Fixed potential calculations must be performed under the implicit solvent model; that is, when sys.fixedP = true, sys.sol must also be set to true.**
> 2. **Currently, fixed potential calculations are only supported when task = scf.**
> 3. **ElectrodePotential_SHE=ElectrodePotential_PZC+ElectrodePotential_SHE(PZC)**

## 2.30 Wannier interpolation band structure calculation

This section will use the $Si$ system as an example to illustrate how to perform interpolated band structure calculations using Wannier functions in DS-PAW.

### 2.30.1 *Si* Interpolation Band Structure Input File

The input files include the parameter file *wannier.in* and the structure file *structure.as*. *wannier.in* is shown below:

```
# task type
task = wannier
sys.structure = structure.as
sys.symmetry = false
sys.functional = PBE
sys.spin = none
cal.methods = 1
cal.smearing = 1
cal.ksamping = G
cal.kpoints = [16,16,16]
cal.totalBands = 12

#wannier related
wannier.functions = 12
wannier.wannMaxIter = 20000
wannier.outStep = 50

#interpolated band related
wannier.interpolatedBand = true
wannier.kpointsLabel= [G,X,W,K,G,L]
wannier.kpointsCoord= [0, 0, 0, 0.5, 0, 0.5, 0.5, 0.25, 0.75, 0.375, 0.375, 0.75, 0, 0,␣
↪0, 0.5, 0.5, 0.5]
wannier.kpointsNumber = [100]

io.charge = true
io.wave = true
```

*wannier.in* Input Parameters:

- `task = wannier`: Sets the calculation type; the new optional value *wannier* enables Wannier function calculations.
- `wannier.functions`: Sets the number of Wannier functions;
- `wannier.wannMaxIter`: Sets the total number of iterations in the process of solving for the maximally localized Wannier functions;
- `wannier.outStep`: Sets the step for outputting iterative results in the output file when task=wannier;
- `wannier.interpolatedBand`: Controls the switch for interpolated band calculation;
- `wannier.kpointsLabel`: Sets the high-symmetry point labels for Wannier function interpolation band fitting.
- `wannier.kpointsCoord`: Sets the high-symmetry point coordinates for Wannier function interpolation band fitting.
- `wannier.kpointsNumber`: Sets the number of k-points between high-symmetry k-points for interpolated band calculation; for example, setting the parameter to `wannier.kpointsNumber = [100]`, the number of k-points between the high-symmetry points G and X is 100, which is used to determine the k-point density; the code performs equally-spaced k-point sampling between high-symmetry points X and W, W and K, K and G, and G and L. The actual number of k-points can be found in the parameter printing section of DS-PAW.log.

The *structure.as* file is referenced as follows:

```
Total number of atoms
2
Lattice
0.00 2.75 2.75
2.75 0.00 2.75
2.75 2.75 0.00
Direct
Si -0.125000000 -0.125000000 -0.125000000
Si 0.125000000 0.125000000 0.125000000
```

**Note**

1. **The initial projection is set in the structure.as file. First, add the WannProj tag on line 7, and then write the initial projection orbital names after the atomic coordinates.** Refer to the Wannier section in *Parameters Explanation* for the recognizable projection orbital names for DS-PAW.
2. **In this example, initial projections are not customized, and the program randomly selects them. If you need to define initial projections, you can refer to the following format.**

The *structure.as* file for custom initial projection orbitals is referenced as follows:

```
Total number of atoms
2
Lattice
0.00 2.75 2.75
2.75 0.00 2.75
2.75 2.75 0.00
Direct WannProj
Si -0.125000000 -0.125000000 -0.125000000  [s,p,sp3-1,sp3-2]
Si 0.125000000 0.125000000 0.125000000     [s,p,sp3-3,sp3-4]
```

**Note**

1. **When customizing initial projection orbitals, the total number of projection orbitals in the structure.as file must equal the number of Wannier functions (wannier.functions), otherwise the program will report an error.**
2. **In this example, the total number of projected orbitals is 2*(1+3+1+1) = 12, which is consistent with the wannier.functions = 12 parameter setting.**
3. **For atoms without the need to set initial projection orbitals, simply write `[]` after the corresponding coordinates.**
4. **In this example, cal.totalBands is set to 12, so 12 cores are required when submitting the calculation.**

### 2.30.2 run the program

Once the input files are ready, upload the *wannier.in* and *structure.as* files to the server and run the *DS-PAW wannier.in* following the method described in structure relaxation.

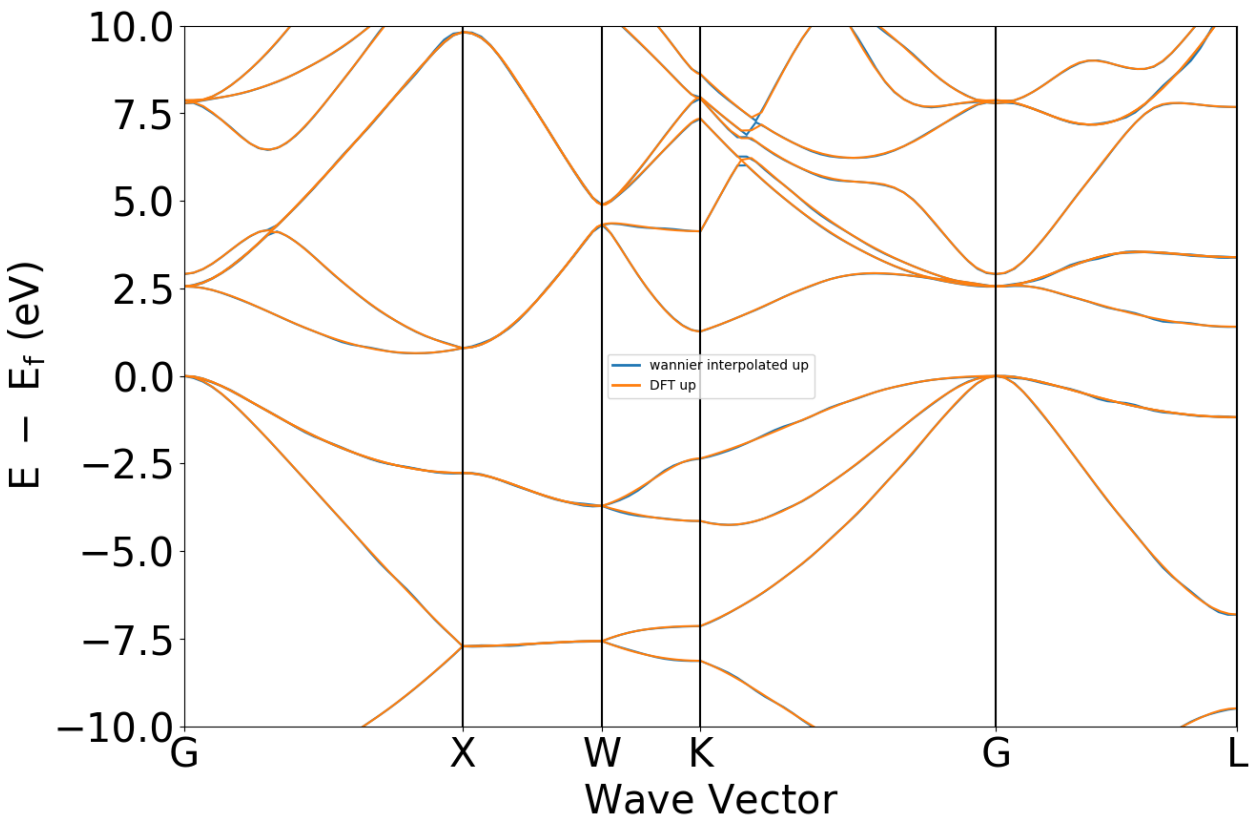
### 2.30.3 Analysis Results Analysis

Based on the input files described above, the calculation will generate output files such as *DS-PAW.log* and *wannier.h5*.

- *DS-PAW.log* : The log file generated after DS-PAW completes the Wannier interpolation band calculation.
- *wannier.h5* : The **h5** output file corresponding to the Wannier function interpolated band structure calculation; for details on the data structure, see the *Output File Format Specification* section;

You can use the *Auxiliary Tool User Guide* –> band data processing –> *bandplot.py* script to directly plot the Wannier interpolated bands, reading the *wannier.h5* file.

You can also use *bandcompare.py* to compare the Wannier interpolated band structure with the DFT band structure. See *Auxiliary Tool User Guide* for specific operations. The band comparison effect should be as follows:




**Note**

1. **Wannier calculations do not support opening pob, and the number of DFT bands calculated (cal.totalBands) changes with the number of cores (cores) used. Therefore, it is recommended that the number of cores used for Wannier calculations be consistent with the cal.totalBands parameter.**
2. **When the number of Wannier functions (wannier.functions) is less than cal.totalBands, the disentanglement process is required during the maximal localization of Wannier functions. In this case, if the user does not define an energy frozen window (wannier.disFrozWin), the program will perform the calculation using the default frozen window.**
3. **If a custom Frozen Window is defined, the number of bands included in `wannier.FrozWin` cannot exceed the number of `wannier.functions`; otherwise, the program will report error E4024. At the same time, the rationality of the window needs to be ensured, or good fitting results cannot be obtained.**
4. **The 2023A version of DS-PAW does not support Wannier calculations with the spin type set to non-collinear.**

## 2.31 ref References

# 3

# Application Cases

This chapter presents various application examples of DS-PAW, including how to calculate magnetic moments and how to calculate antiferromagnetic materials. Users can gain a deeper understanding of DS-PAW software through the following application tutorials.

## 3.1 Calculation of the Magnetic Moment of Atom $O$

This section introduces the calculation of magnetic systems using a single oxygen atom as an example.

### 3.1.1 File Preparation for Self-Consistent Calculation of an $O$ Atom

Since this calculation involves the magnetic moment of a single oxygen atom, structural relaxation is not necessary. We proceed directly to the self-consistent field (SCF) calculation. Prepare the input files *scf.in* and *structure.as*. *scf.in* is as follows:

```
task = scf
sys.symmetry = false
sys.structure = structure.as
sys.spin = collinear
cal.smearing = 1
cal.sigma = 0.01
cal.kpoints = [1, 1, 1]
```

The following parameters in the input file are crucial for this calculation:

- `sys.symmetry`: DS-PAW can reduce computational cost by using symmetry, but it may also lead to unreasonable results such as energy degeneracy. Symmetry is turned off in this calculation;
- `sys.spin`: Specifies the systems magnetism as **collinear**.
- `cal.kpoints`: For non-periodic dimensions, the k-point can be set to 1;

The *structure.as* file is referenced as follows:

```
Total number of atoms
1
Lattice
7.50000000 0.00000000 0.00000000
0.00000000 8.00000000 0.00000000
0.00000000 0.00000000 8.90000000
Cartesian
O  0.00000000 0.00000000 0.00000000
```

The structure file uses Cartesian coordinates, hence the coordinate type in line 7 is `Cartesian`; to minimize symmetry in the structure, the lattice was modified to a **[7.5, 8, 8.9]** lattice.

### 3.1.2 Run the program

After preparing the input files, upload the *scf.in* and *structure.as* files to the environment where DS-PAW is installed, and run the *DS-PAW scf.in* command.

### 3.1.3 Analysis of calculation results.

After the calculation based on the input files is completed, output files such as *DS-PAW.log* and *scf.h5* will be generated.

Open the *scf.h5* file with HDFView, and the Eigenvalue data will be as follows:

```
▼ object {6}
  ▶ AtomInfo {5}
  ▶ Eigenvalue {3}
  ▶ Energy {3}
  ▶ Force {1}
  ▼ MagInfo {1}
    ▼ TotalMag [1]
        0 : 2.000996884905
  ▶ Stress {2}
```

The number of up-spin electrons is 4 and the number of down-spin electrons is 2, obtained from the **Eigenvalue - Spin - Occupation** section of *scf.h5*. The total magnetic moment is $2\mu B$, obtained from the **MagInfo** section of *scf.h5*, and also confirmed as $2\mu B$ in *DS-PAW.log*.

## 3.2 $NiO$ antiferromagnetic calculation

This section will introduce how to set up antiferromagnetic calculations using the NiO system as an example.

### 3.2.1 Self-consistent calculation for the $NiO$ system

This case omits the structure relaxation process; users should perform a structure relaxation calculation first when reproducing this case. Prepare the parameter file *scf.in* and the structure file *structure.as*, and the *scf.in* file is as follows:

```
task = scf
sys.structure = structure.as
sys.spin = collinear
cal.smearing = 4
cal.kpoints = [8, 8, 8]
cal.cutoff = 650
```

The following parameters in the input file of this calculation require special attention:

- `cal.smearing`: The **tetrahedron method with Bloechl correction** is employed in this calculation, and `sigma` will be forced to **0** when using this method.
- `sys.spin`: Specifies the magnetism of the system. **NiO** is an antiferromagnetic material, so the magnetism is set to `collinear`;
- `cal.cutoff`: Sets the plane-wave cutoff to **650 eV**.

Refer to the *structure.as* file as follows:

```
Total number of atoms
4
Lattice
4.16840000 2.08420000 2.08420000
2.08420000 4.16840000 2.08420000
2.08420000 2.08420000 4.16840000
Cartesian Mag
Ni 1.04210000 1.04210000 1.04210000   2.0
Ni 5.21050000 5.21050000 5.21050000   -2.0
O 3.12630000 3.12630000 3.12630000    0
O 7.29470000 7.29470000 7.29470000    0
```

To set the magnetic moments, add the **Mag** tag after `Cartesian` on the seventh line of the structure file. Then, set the magnetic moment for each atom on the line containing its coordinates. Because we need to represent antiferromagnetism (the entire system does not show a net magnetic moment, but individual atoms have magnetic moments), this example uses a unit cell of 4 atoms. The magnetic moments for the 4 Ni atoms are set to **2, -2, 0, 0**.

**Note**

1. The **Mag** tag allows setting the magnetic moments for each atom in the system. For collinear spin calculations, the total magnetic moment of each atom can be added. For spin-orbit coupling calculations, the magnetic moments in the x, y, and z directions need to be added using the tags **Mag_x**, **Mag_y**, and **Mag_z**. Add the magnetic moments in the three directions after the corresponding atomic coordinates. Taking the NiO system as an example, if a spin-orbit coupling calculation is performed, the magnetic moment settings should be as follows:

```
Total number of atoms
4
Lattice
4.16840000 2.08420000 2.08420000
2.08420000 4.16840000 2.08420000
```

(continues on next page)

(continued from previous page)

```
2.08420000 2.08420000 4.16840000
Cartesian Mag_x Mag_y Mag_z
Ni 1.04210000 1.04210000 1.04210000   0.0 0.0 2.0
Ni 5.21050000 5.21050000 5.21050000   0.0 0.0 -2.0
O 3.12630000 3.12630000 3.12630000    0.0 0.0 0.0
O 7.29470000 7.29470000 7.29470000    0.0 0.0 0.0
```

### 3.2.2 run the program

After preparing the input files, upload the *:guilabel: `scf.in` `* and *:guilabel: `structure.as` `* files to the environment where DS-PAW is installed, and run the command *:guilabel: `DS-PAW scf.in` `*.

### 3.2.3 Analysis of self-consistent field (SCF) calculation results

Based on the input file described above, after the calculation is completed, the following output files will be generated: *DS-PAW.log* and *scf.h5*, etc. From *DS-PAW.log*, the total magnetic moment after the self-consistent calculation can be read as $1e - 8\mu B$, which is almost 0.

### 3.2.4 $NiO$ Density of States Calculation

After that, we will prepare for the density of states (DOS) calculation, preparing the parameter file *pdos.in*, the structure file *structure.as*, and the charge density file *rho.bin* obtained from the self-consistent calculation. The *pdos.in* file is as follows:

```
task = dos
sys.structure = structure.as
sys.spin = collinear
cal.iniCharge = ./rho.bin
cal.smearing = 4
cal.kpoints = [16, 16, 16]
cal.cutoff = 650
dos.range = [-20, 20]
dos.resolution = 0.05
dos.project = true
```

*pdos.in* Input Parameters:

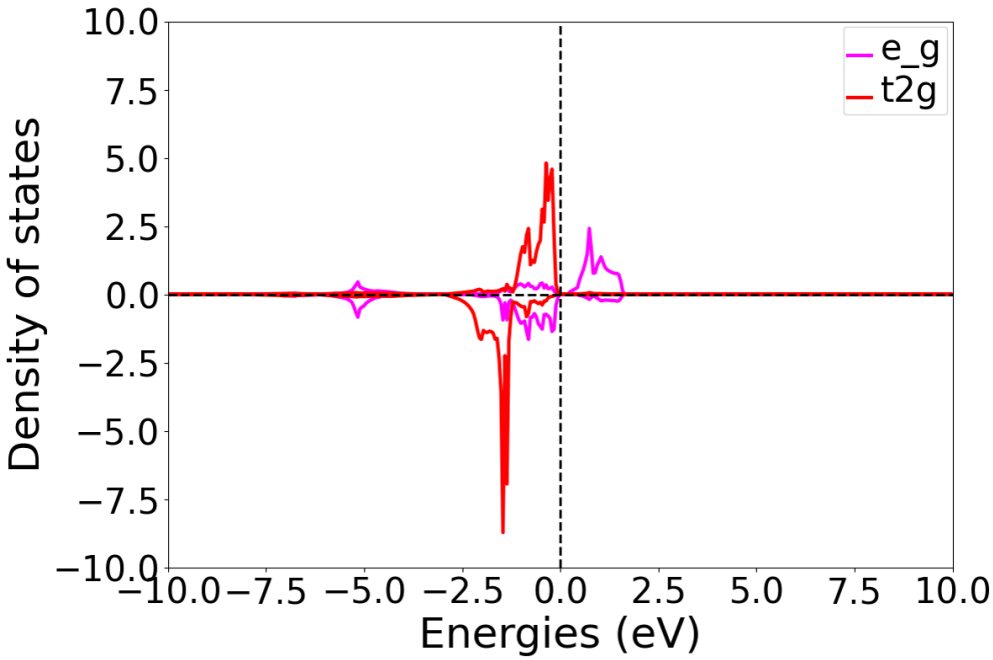
- `dos.range`: Specifies the energy range for DOS calculation, from -20 to 20 eV.
- `dos.resolution` : indicates the interval precision for sampling within the energy calculation range;
- `dos.project`: Controls the projection calculation for the density of states; projection for the density of states is enabled in this calculation.

### 3.2.5 run the program

Upload the newly created *pdos.in* file to the server, and then run the command *DS-PAW pdos.in*.

### 3.2.6 Analysis of DOS (Density of States) Calculation Results

After completing the calculation based on the input file, output files such as *DS-PAW.log* and *dos.h5* will be generated. Using the relevant scripts in *Auxiliary Tool User Guide* to process the *dos.h5* file and analyze the t2g and eg orbitals of the 2nd Ni atom, the density of states distribution shown below is obtained. This is the result without a U value applied:




### 3.2.7 $NiO$ system DFT+U density of states calculation

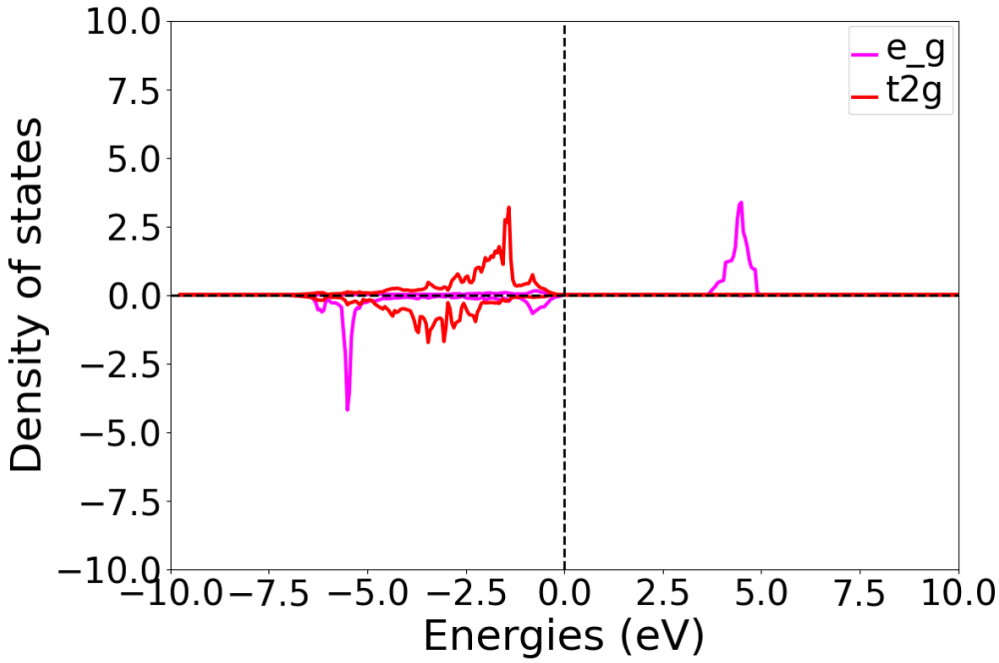
The calculation procedure for the density of states (DOS) of the NiO system using DFT+U is the same as that described in the previous section for the NiO system, with the difference being that DFT+U parameters need to be included in both the self-consistent field (SCF) calculation and the DOS calculation. The input parameters that need to be added are as follows:

```
#correction related
corr.dftu=true
corr.dftuForm = 1
corr.dftuElements =[Ni]
corr.dftuOrbital=[d]
corr.dftuU = [8]
corr.dftuJ = [0.95]
```

Here are a few parameters in the input file that require special attention for this calculation:

- `corr.dftu` sets the switch for turning on DFT+U, which is set to true in this example;
- `corr.dftuForm` sets the DFT+U method, with 1 corresponding to the DFT+U+J method (Liechtensteins formulation);
- `corr.dftuElements` sets the elements to which U is applied, which is Ni in this example;
- `corr.dftuOrbital` specifies the orbitals to which the U correction is applied, which is set to d orbitals in this example;
- `corr.dftuU` sets the specific U value, which is set to 8 in this example;
- `corr.dftuJ` sets the specific J value, which is set to 0.95 in this example;

After the self-consistent field (SCF) and density of states (DOS) calculations are completed, the distribution of the t2g and eg orbital density of states for the second Ni atom after the DFT+U calculation is analyzed. The resulting distribution plot is shown below:




**Note**

1. **DFT+U allows setting U values for multiple elements and their corresponding orbitals. For example, to set U=8 and J=0.95 for Nis d orbitals, and U=1 and J=0 for Os p orbitals, the settings are as follows: corr.dftuElements = [Ni,O] corr.dftuOrbital = [d,p] corr.dftuU = [8,1] corr.dftuJ = [0.95,0].**
2. **The default DFT+U method is DFT+U (Dudarevs formulation), corresponding to the parameter corr.dftuForm = 2. When using this method, the J value is forced to be 0, so setting the J value is invalid in this case.**

## 3.3 $AuAl$ slab model work function calculation

This section will demonstrate how to calculate the work function using the $AuAl$ slab model as an example.

### 3.3.1 File Preparation for Self-Consistent Calculation of the $AuAl$ Slab Model

This case omits the structure relaxation process; users need to perform structure relaxation calculations before reproducing this case. Prepare the parameter file *scf.in* and the structure file *structure.as*. The *scf.in* file is as follows:

```
task = scf
sys.structure = structure.as
sys.spin = collinear
cal.smearing = 4
cal.kpoints = [8, 8, 1]
cal.cutoff = 530

io.potential=true
potential.type = hartree

#correction related
```

(continues on next page)

(continued from previous page)

```
corr.dipol = true
corr.dipolDirection = c
```

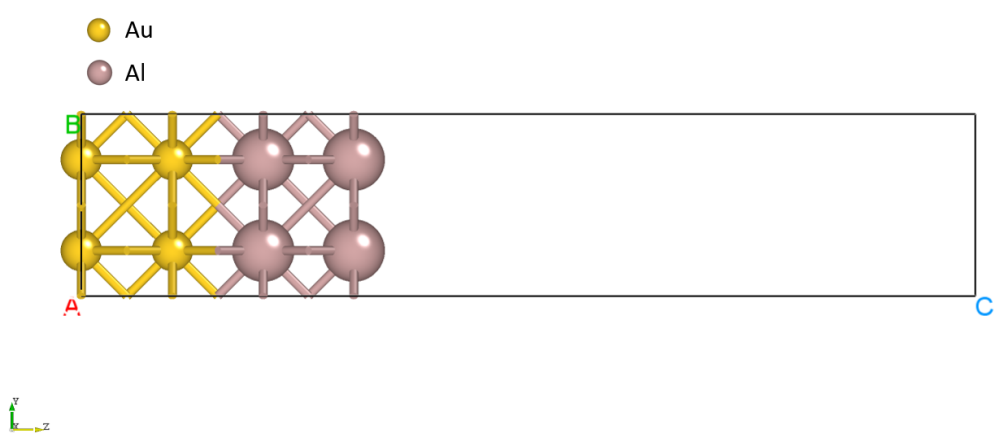
The following parameters in the input file of this calculation require special attention:

- `io.potential` is the switch for calculating the potential function in the self-consistent field (SCF) calculation;
- `potential.type` controls the type of potential function to be saved. The electrostatic potential data is needed when calculating the work function, and here we set potential.type = hartree;
- `corr.dipol` is the switch for dipole correction; set to true in this example;
- `corr.dipolDirection` In this example, the direction of the dipole correction is set to the c direction of the lattice vectors.

The *structure.as* file is referenced as follows:

```
Total number of atoms
8
Lattice
 4.06384898 0.00000000 0.00000000
 0.00000000 4.06384898 0.00000000
 0.00000000 0.00000000 20.00000000
Cartesian
Au 1.01596223 1.01596223 0.00000000
Au 3.04788672 3.04788672 0.00000000
Au 3.04788672 1.01596224 2.03914999
Au 1.01596224 3.04788672 2.03914999
Al 1.01596224 1.01596224 4.07109999
Al 3.04788673 3.04788673 4.07109999
Al 3.04788673 1.01596224 6.09585000
Al 1.01596224 3.04788673 6.09585000
```

The structure is shown in the figure below:



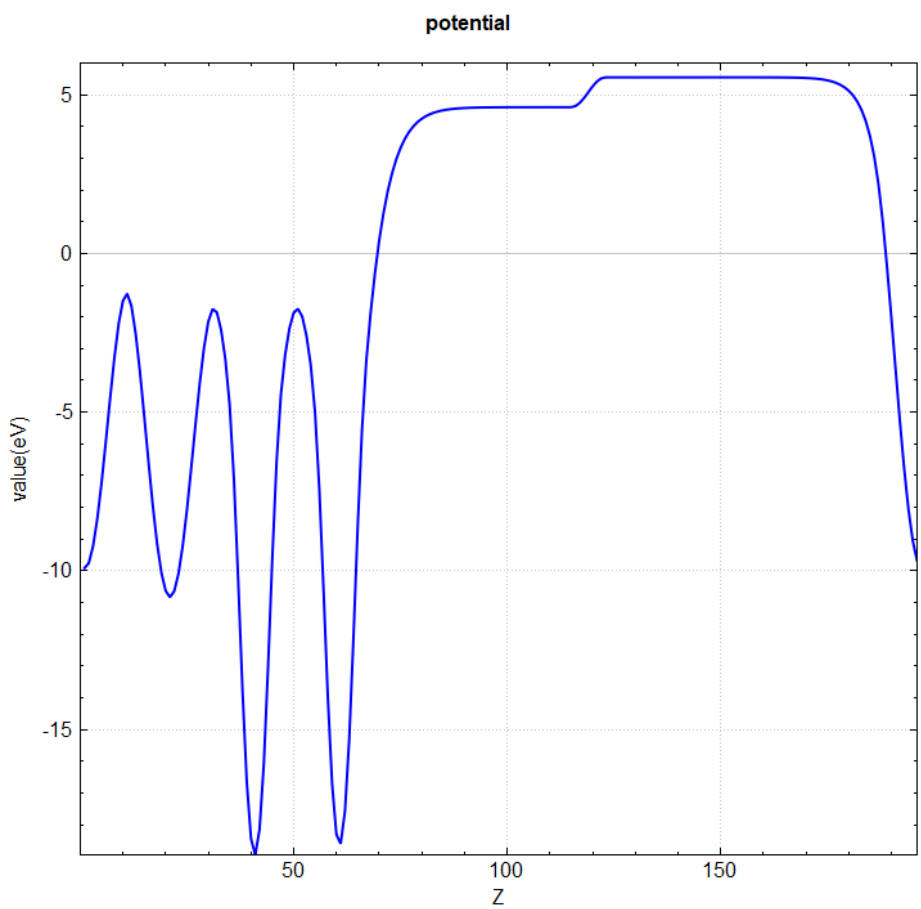
### 3.3.2 run program execution

After preparing the input files, upload the *scf.in* and *structure.as* files to the environment where DS-PAW is installed and run the command *DS-PAW scf.in*.

### 3.3.3 workfunction data analysis

After the calculation based on the above input files, the output files such as *DS-PAW.log* and *scf.h5* will be generated. Processing the data from *scf.h5* yields the work function.

You can use a **python** script to analyze the *scf.h5* file, averaging the 3D potential function in-plane. See the *Auxiliary Tool User Guide* section for specific instructions. The resulting vacuum direction potential curve is shown below:




From the in-plane averaged potential plot, the vacuum potentials for Au and Al are 5.5 eV and 4.6 eV, respectively.

The Fermi level can be read from *scf.h5* as 0.113 eV.

Based on the formula $w = -e\phi - E_F$, the work function of Au in the $AuAl$ slab model is **5.387** eV, and that of Al is **4.487** eV. The literature values[1] are: Aus work function in the range of **5.10-5.47** eV, and Als work function in the range of **4.06-4.26** eV.

[1] WilliamăM Haynes, DavidăR Lide, and ThomasăJ Bruno. *CRC handbook of chemistry and physics*. CRC press, 2016. doi:10.1201/9781315380476.

## 3.4 $Ru - N4$ Computational Electrocatalysis of Nitrogen Reduction Reaction

This section will demonstrate how to simulate an electrocatalytic nitrogen reduction reaction (eNRR) using DS-PAW. The reaction uses a carbon-based supported transition metal Ru single atom as a catalyst, and DS-PAW is used to simulate the adsorption and reduction process of nitrogen molecules.

During electrochemical interfacial reactions, the interface is typically connected to an external electrode with a constant electrode potential. To ensure that the electronic chemical potential equilibrates with the external electrode potential, meaning the grand canonical ensemble for electrons, there will be an influx and outflux of electrons in the actual system. Traditional first-principles calculations are usually performed under the canonical ensemble, i.e., under the condition of charge conservation, and thus cannot accurately describe electrochemical interfacial reactions. We will refer to the calculation model expanded under charge conservation as the constant charge model (CCM).

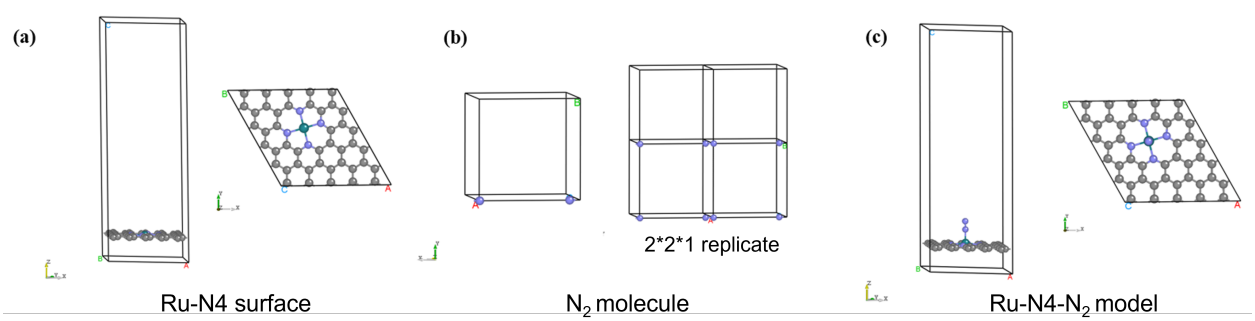
Since the constant charge model is not suitable for handling electrochemical interface problems, we can adopt first-principles calculations expanded in the electronic grand canonical ensemble. This calculation method is also known as the fixed potential method/constant potential method. In this case, we will refer to the fixed potential calculation model as the constant potential model (CPM).

### 3.4.1 Flow Calculation Procedure and Input Files

This example simulates the adsorption and reduction of nitrogen molecules on a carbon-supported transition metal Ru single atom catalyst using DS-PAW. The simulated reaction is the adsorption of nitrogen molecules on a carbon-supported Ru single atom, which can be simplified as: $(Ru - N_4) + N_2 = (Ru - N_4 - N_2)$. Three different models, CCM_vacuum, CCM_water, and CPM_water, are used in the calculation. The entire calculation procedure can be roughly divided into four steps, detailed as follows:

#### 3.4.1.1 Build the model

The models include: (a) a carbon-supported Ru atom model ($Ru - N_4$), (b) a single $N_2$ molecule model, and (c) a carbon-supported Ru atom model with adsorbed $N_2$ molecule ($Ru - N_4 - N_2$). The model structures are shown below:




#### 3.4.1.2 Relaxation Structure Relaxation

Perform structural relaxation on the constructed structure to obtain a stable structure. Core parameters required for structural relaxation in DS-PAW:

```
relax.max =  200                  # Specify the maximum number of ionic steps for␣
↪structure relaxation
relax.freedom = atom              #Specify the degree of freedom for structural␣
↪relaxation
relax.convergence =  0.02         #Specifies the convergence criterion for atomic forces␣
↪during structural relaxation
relax.methods = CG                # Conjugate gradient method is specified for␣
↪structural relaxation.
```

#### 3.4.1.3 Energy Calculation

Energy calculations were performed under different model conditions to obtain the energies of stable configurations. The results are presented below, categorized by model.

##### CCM_vacuum

Standard scf calculation under vacuum layer to obtain the energy under the CCM_vacuum model. The core parameters required for a **single point energy calculation** in DS-PAW are listed below:

```
task = scf

# Self-consistent field calculation related
cal.methods = 1                   #Specifies the self-consistent electronic part␣
↪optimization method as BD (block Davidson) method
cal.smearing = 1                  # Specifies Gaussian smearing to set the fractional␣
↪occupation of each wavefunction
cal.k_sampling = G                  #Specifies the method for generating the Brillouin␣
↪zone k-point grid
cal.kpoints = [2, 2, 1]           #Specifies the size of the k-point grid sampling in␣
↪the Brillouin zone

cal.cutoffFactor = 1.0            # Specifies the coefficient for the plane-wave basis␣
↪cutoff energy parameter cal.cutoff


corr.VDW = true                   # Specifies that van der Waals correction calculation␣
↪is enabled.
corr.VDWType = D2G                #Specifies the Grimme's DFT-D2 method for van der Waals␣
↪correction
```

##### CCM_water

In the CCM model, implicit solvation models can also be used to consider solvent effects. Here, we take an aqueous solution as an example and list the core parameters that need to be set to introduce the **solvation model** in the SCF calculation using DS-PAW:

```
task = scf

# Solvation model related
sys.sol = true                        # Specify whether to apply the implicit solvation␣
↪model, true means enabled.
sys.solEpsilon = 78.4                 # Specify the dielectric constant of the solvent,␣
```

(continues on next page)

(continued from previous page)

```
↪78.4 for water
sys.solLambdaD = 3.04                  # Specify the Debye length in the Poisson-
↪Boltzmann equation, in Å
sys.solTAU = 0                         # Specify the magnitude of the effective␣
↪interfacial tension per unit area, in eV/Å^2
io.boundCharge = false                 # Specify whether to output the solvent-bound␣
↪charge density file, false means off.
```

#### CPM_water

In DS-PAW, the energy under the CPM model can be obtained by using the fixed potential method. In the newly released 2023A version, the fixed potential calculation must introduce a solvation model. Here are the core parameters for **fixed potential calculations** in an implicit aqueous environment using DS-PAW:

```
task = scf
# Solvent model related
sys.sol = true                         # Specify whether to apply the implicit solvation␣
↪model, true means enabled.
sys.solEpsilon = 78.4                  # Specifies the dielectric constant of the␣
↪solvent, with a default value of 78.4 for water.
sys.solLambdaD = 3.04                  #Specifies the Debye length in the Poisson-
↪Boltzmann equation, in Å
sys.solTAU = 0                         # Specifies the effective interfacial tension per␣
↪unit area, in eV/Å^2
io.boundCharge = false                 # Specifies whether to output the solvent bound␣
↪charge density file, false means off

# Related to fixed potential settings
sys.fixedP = true                      # Specifies whether to enable fixed potential␣
↪calculation, true means enabled
sys.fixedPConvergence = 0.01           # Specifies the convergence criterion for the␣
↪fixed potential calculation. The calculation ends when the delta_electron between two␣
↪self-consistent calculations is less than the convergence criterion.
sys.fixedPMaxIter = 60                 #Specify the maximum number of iterations for␣
↪fixed potential calculation.
sys.fixedPPotential = 0                # Specifies the target electrode potential value␣
↪for the fixed potential calculation, using SHE as the reference electrode potential by␣
↪default.
sys.fixedPType = SHE                   # Specifies the potential type for the potential␣
↪value given by sys.fixedPPotential, supporting SHE (Standard Hydrogen Electrode) and␣
↪PZC (Potential of Zero Charge)
```

#### 3.4.1.4 ReactionEnergy Calculation

**This example selects three different computational models. First, the adsorption reaction equations for each model are introduced:**

#### CCM_vacuum

In this model, the adsorption reaction can be written as:

$(Ru - N4) + N2(idealgas) = (Ru - N4 - N2)$

We define $\Delta E$ as the reaction energy, and the calculation expression for the reaction energy is:

$\Delta E = E0(Ru - N4 - N2) - E0(Ru - N4) - E0(N2)$

where E0 corresponds to the total energy of the system in the vacuum model ( $\sigma \rightarrow 0$), which can be obtained from the self-consistent calculation results in the *scf.h5* or *system.json* file by searching for the keyword TotalEnergy0.

### CCM_water

Under this model, the adsorption reaction equation can be written as:

$(Ru - N4)(inwater) + N2(idealgas) = (Ru - N4 - N2)(inwater)$

$\Delta E = E0(Ru - N4 - N2) - E0(Ru - N4) - E0(N2)$

Where, E0 corresponds to the total energy of the system under the model of aqueous solution immersion ( $\sigma \rightarrow 0$), this value can be obtained from the self-consistent field calculation, specifically from the *scf.h5* or *system.json* file, by searching for the keyword TotalEnergy0.

### CPM_water

The reaction process simulated in this model is the adsorption of $N_2$ in the gas phase on a catalyst surface wetted by an aqueous solution and in contact with a 0V vs. SHE (Standard Hydrogen Electrode) electrode. **Two different adsorption reaction equations can be written for this process.** For ease of description, we **define the following physical quantity symbols**:

- $ne0$ : Number of core electrons in the neutral system.
- $ne$ : Total number of electrons in the system when the system voltage is set (value set by sys.fixedPPotential, corresponding to 0 V in this example)
- $dne$ : The charge of the system at the set system voltage: $dne = ne - ne0$
- $\mu_e$ : Electronic chemical potential of the system, with the potential zero point at the bulk solution (i.e., the potential value at the lowest charge density point obtained from DFT calculations)
- $\Delta e$ : Difference in the number of valence electrons between the adsorbed system (eAB) and the sum of valence electrons of the substrate and adsorbate (eA+eB).
- $\Omega 0$ : grand total energy(sigma→0): the total energy of the system in the grand canonical ensemble of electrons, **which is expressed as**: $\Omega 0 = E0 - dne * \mu_e$

**In this case, the adsorption reaction formula under the CPM_water model can be written as follows:**

**Method 1**, considering $\Delta e$ in the reaction equation, the adsorption reaction can be written as:

$Ru - N4(0Vvs.SHE) + N2(idealgas) = Ru - N4 - N2(0Vvs.SHE) - \Delta e$

$\Delta E = E0(Ru - N4 - N2) - \Delta e * \mu_e - E0(Ru - N4) - E0(N2)$

The values of E0 can be obtained from the self-consistent calculation results in the *scf.h5* or *system.json* file, by searching for the keyword TotalEnergy0.

The values of $ne$ and $\mu_e$ can be obtained from the self-consistent calculation results in the *DS-PAW.log* file (or *scf.h5* or *system.json*), by searching for the keywords Electron and Chemical Potential(electron) under the last LOOP.

**Method two**, considering the total energy of the system $\Omega 0$ in the grand canonical ensemble of electrons.

Since the constant potential calculation simulates the electronic grand canonical ensemble, the total energy $E0$ in the reaction energy calculation should be replaced by $\Omega 0$. The adsorption reaction equation can be written as:

$(Ru - N4)(0Vvs.SHE) + N2(idealgas) = (Ru - N4 - N2)(0Vvs.SHE)$

$\Delta E = \Omega 0(Ru - N4 - N2) - \Omega 0(Ru - N4) - \Omega 0(N2)$

Where the value of $\Omega 0$ can be obtained from the self-consistent calculation in the *DS-PAW.log* file (or the *scf.h5* or *system.json* files), by searching for the keyword Grand Total Energy in the last LOOP.

**Since the potential of (Ru-N4) versus (Ru-N4-N2) is 0V vs. SHE, a fixed potential calculation at 0V is performed for (Ru-N4) and (Ru-N4-N2). Data is extracted from the corresponding output files of DS-PAW, and the following table is obtained, with energy units in eV:**

| system | $E0$ | $nE0$ | $ne$ | $dne$ | $\mu_e$ | $\Omega 0$ |
|---|---|---|---|---|---|---|
| N2 | -545.9393 | 10 | / | / | / | -545.9393 |
| Ru-N4 | -10572.2452 | 212 | 211.224 | -0.776 | -4.60223 | -10575.81654 |
| Ru-N4-N2 | -11119.6117 | 222 | 221.229 | -0.771 | -4.60054 | -11123.15868 |

Next, we substitute the data from **Table 1** into the corresponding expressions for calculation:

**Method 1** Considering $\Delta e$ in the reaction equation, the reaction energy calculation process is as follows:

$Ru - N4(0Vvs.SHE) + N2(idealgas) = Ru - N4 - N2(0Vvs.SHE) - \Delta e$

$\Delta E = E0(Ru - N4 - N2) - \Delta e * \mu_e - E0(Ru - N4) - E0(N2)$

$= -11119.6117 - (221.229 - 211.224 - 10) * (-4.600) - (-10572.2452) - (-545.9393)$

$= -1.4042eV$

**Method 2** Consider the grand canonical potential $\Omega 0$ of the system, and the reaction energy calculation is as follows:

$(Ru - N4)(0Vvs.SHE) + N2(idealgas) = (Ru - N4 - N2)(0Vvs.SHE)$

$\Delta E = \Omega 0(Ru - N4 - N2) - \Omega 0(Ru - N4) - \Omega 0(N2)$

$= -11123.1586 - (-10575.8165) - (-545.9393)$

$= -1.4027eV$

The calculated adsorption energies obtained using the two methods are consistent. It is apparent that the reaction energy under a fixed potential can be easily computed using $\Omega 0$ as defined in DS-PAW.

### 3.4.2 Run the program

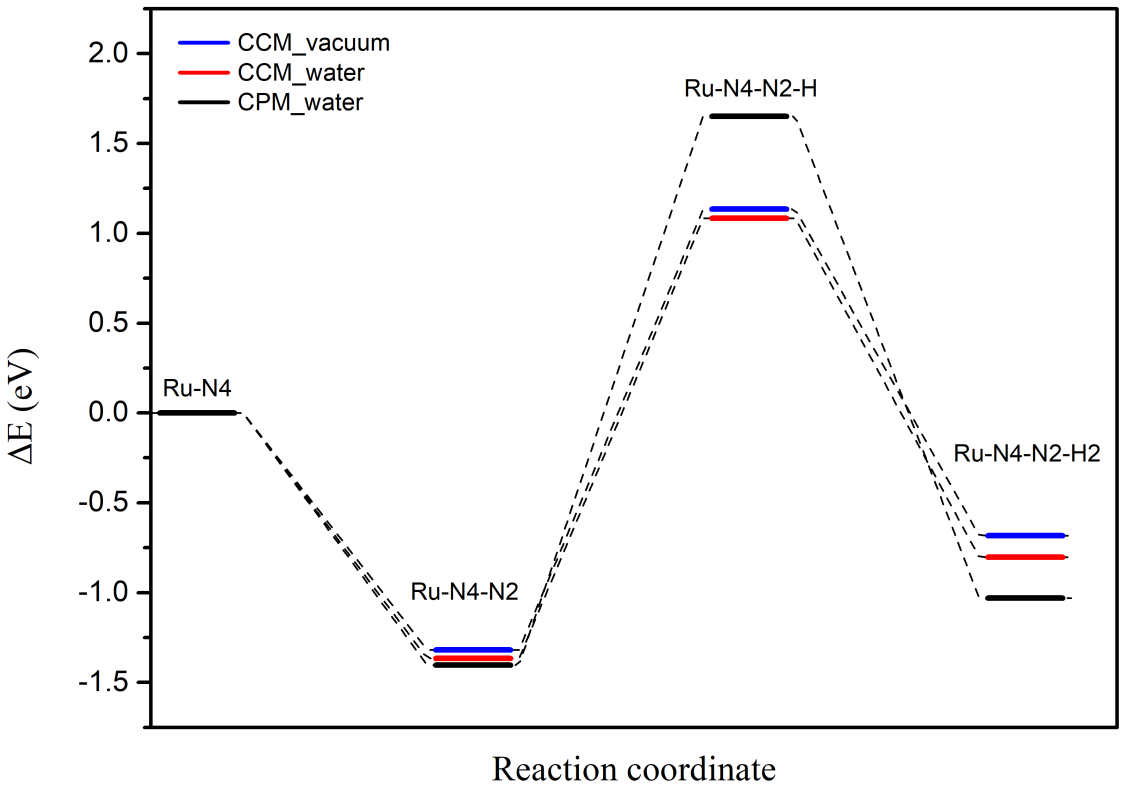
After preparing the input files, upload the in files for structural relaxation calculations, energy calculations, and fixed potential calculations, along with the *structure.as* structure file, to an environment with DS-PAW installed. Following the workflow, run the *DS-PAW input.in* command in multiple steps or submit job scripts to complete multiple calculations.

### 3.4.3 ReactionEnergy Reaction Energy data analysis

Substituting the data from Table 1 into the adsorption reaction equations for the **CCM_vacuum**, **CCM_water**, and **CPM_water** models, we calculate the **reaction energies of the first three steps of the eNRR** for the three models. The results are shown in **Table 2**:

| reaction/ $\Delta e$ (eV) | CCM_vacuum | CCM_water | CPM_water |
|---|---|---|---|
| $(Ru - N4) + N2 = (Ru - N4 - N2)$ | -1.3180 | -1.3668 | -1.4027 |
| $(Ru - N4 - N2) + 0.5H2 = (Ru - N4 - N2 - H)$ | 1.1355 | 1.0833 | 1.6511 |
| $(Ru - N4 - N2 - H) + 0.5H2 = (Ru - N4 - N2 - H2)$ | -0.6833 | -0.8030 | -1.0305 |

The results are then plotted as a reaction coordinate curve, as shown in **Figure 3**:




## 3.5 ref References

# 4

# Pseudopotential Explanation

DS-PAW currently supports three formats of pseudopotentials: `.paw`, `.potcar`, and `.pawpsp`. Users can specify the pseudopotential type using the `sys.pseudoType` parameter.

## 4.1 hzw internal PAW pseudopotential

The DS-PAW defaults to using the **hzw** pseudopotentials (.paw), with the corresponding parameter `sys.pseudoType` set to **-1**. In this case, DS-PAW will read the pseudopotential files from the installation path **/pseudopotential**. Currently, the **hzw** pseudopotential library contains 72 elements, covering elements **1-86** in the periodic table (lanthanides are currently supported only up to Lanthanum).

Regarding the accuracy of the **hzw** pseudopotential: the quick start guide and application examples perform calculations based on multiple functionalities, and the results are in good agreement with the literature, which validates that the **hzw** pseudopotential exhibits high accuracy in various functional calculations.

Furthermore, for the 72 elements in the pseudopotential library, calculations of *equation of state fitting to obtain the equilibrium cell volume* were performed based on the 1.0 version pseudopotentials for the corresponding elemental solids. The test objects included the 72 elements LDA and PBE functional pseudopotentials, totaling 144 pseudopotential files. The calculated volumes were compared with those obtained from the quantum chemistry software **WIEN2k**, and the calculation errors are displayed in the periodic table as follows:

LDA

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H 0.05% | | | | | | | | | | | | | | | | | He 0.03% |
| Li 0.03% | Be 0.09% | | | | | | | | | | | B 0.04% | C 0.03% | N 0.01% | O 0.12% | F 0.11% | Ne 0.14% |
| Na 0.11% | Mg 0.10% | | | | | | | | | | | Al 0.10% | Si 0.04% | P 0.08% | S 0.17% | Cl 0.10% | Ar 0.12% |
| K 0.07% | Ca 0.14% | Sc 0.17% | Ti 0.17% | V 0.24% | Cr 0.43% | Mn 0.57% | Fe 0.35% | Co 0.77% | Ni 1.09% | Cu 1.78% | Zn 2.58% | Ga 0.58% | Ge 0.11% | As 0.04% | Se 0.04% | Br 0.05% | Kr 0.06% |
| Rb 0.12% | Sr 0.21% | Y 0.12% | Zr 0.04% | Nb 0.12% | Mo 0.59% | Tc 0.05% | Ru 0.20% | Rh 0.07% | Pd 0.32% | Ag 0.14% | Cd 0.11% | In 0.35% | Sn 0.15% | Sb 0.07% | Te 0.27% | I 0.16% | Xe 0.00% |
| Cs 0.06% | Ba 0.24% | | Hf 0.13% | Ta 0.14% | W 0.08% | Re 0.12% | Os 0.54% | Ir 0.44% | Pt 0.69% | Au 0.50% | Hg 0.39% | Tl 0.01% | Pb 0.01% | Bi 0.01% | Po 0.21% | At | Rn 0.03% |
| Fr | Ra | | Rf | Db | Sg | Bh | Hs | Mt | Ds | Rg | Cn | Nh | Fl | Mc | Lv | Ts | Og |
| | | | La | Ce | Pr | Nd | Pm | Sm | Eu | Gd | Tb | Dy | Ho | Er | Tm | Yb | Lu |
| | | | Ac | Th | Pa | U | Np | Pu | Am | Cm | Bk | Cf | Es | Fm | Md | No | Lr |

PBE

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H 0.03% | | | | | | | | | | | | | | | | | He 0.01% |
| Li 0.07% | Be 0.09% | | | | | | | | | | | B 0.02% | C 0.09% | N 0.11% | O 0.09% | F 0.06% | Ne 0.15% |
| Na 0.20% | Mg 0.12% | | | | | | | | | | | Al 0.10% | Si 0.05% | P 0.16% | S 0.09% | Cl 0.03% | Ar 0.03% |
| K 0.12% | Ca 0.05% | Sc 0.14% | Ti 0.17% | V 0.22% | Cr 0.53% | Mn 0.86% | Fe 0.97% | Co 0.80% | Ni 1.13% | Cu 2.39% | Zn 3.31% | Ga 0.04% | Ge 0.04% | As 0.03% | Se 0.02% | Br 0.02% | Kr 0.20% |
| Rb 0.18% | Sr 0.31% | Y 0.11% | Zr 0.02% | Nb 0.05% | Mo 0.55% | Tc 0.03% | Ru 0.26% | Rh 0.06% | Pd 0.72% | Ag 0.06% | Cd 0.09% | In 0.18% | Sn 0.08% | Sb 0.01% | Te 0.27% | I 0.18% | Xe 0.07% |
| Cs 0.08% | Ba 0.29% | | Hf 0.01% | Ta 0.01% | W 0.03% | Re 0.02% | Os 0.36% | Ir 0.32% | Pt 0.50% | Au 0.31% | Hg 0.85% | Tl 0.03% | Pb 0.16% | Bi 0.13% | Po 0.34% | At | Rn 0.15% |
| Fr | Ra | | Rf | Db | Sg | Bh | Hs | Mt | Ds | Rg | Cn | Nh | Fl | Mc | Lv | Ts | Og |
| | | | La | Ce | Pr | Nd | Pm | Sm | Eu | Gd | Tb | Dy | Ho | Er | Tm | Yb | Lu |
| | | | Ac | Th | Pa | U | Np | Pu | Am | Cm | Bk | Cf | Es | Fm | Md | No | Lr |

**WIEN2k data source**: https://github.com/abinit/pseudo_dojo

**Since WIEN2k website does not provide calculation data for La and At, the comparison data for La and At are not shown in the table.**

By comparison, the equilibrium volume obtained by fitting the equation of state in version 1.0 is basically consistent with the results from **WIEN2k** software. The largest errors are for the element Zn, with errors of **2.58%** and **3.31%** for the LDA and PBE pseudopotentials, respectively. Optimization of the pseudopotentials for these two elements is underway. The errors for the remaining elements are basically controlled within **0.1%**, which further validates the overall accuracy of the pseudopotential library.

The 1.1 version of the pseudopotentials will be released on 2024/12/31. Performance data will be announced shortly, stay tuned

## 4.2 VASP Pseudopotentials

DS-PAW provides an interface for using external POTCAR formatted pseudopotentials (.potcar), with the corresponding parameter `sys.pseudoType` set to **10**. In this case, DS-PAW will read the pseudopotential files from the default path **./**. **Due to copyright restrictions, DS-PAW only provides the interface for using VASP pseudopotentials, and the pseudopotential files must be prepared by the user.** When using this feature, users need to modify the pseudopotential filenames accordingly. For example, if using the LDA pseudopotential for silicon, the corresponding POTCAR should be renamed to Si_LDA.potcar and placed in the specified directory (which can be set via sys.pseudoPath).

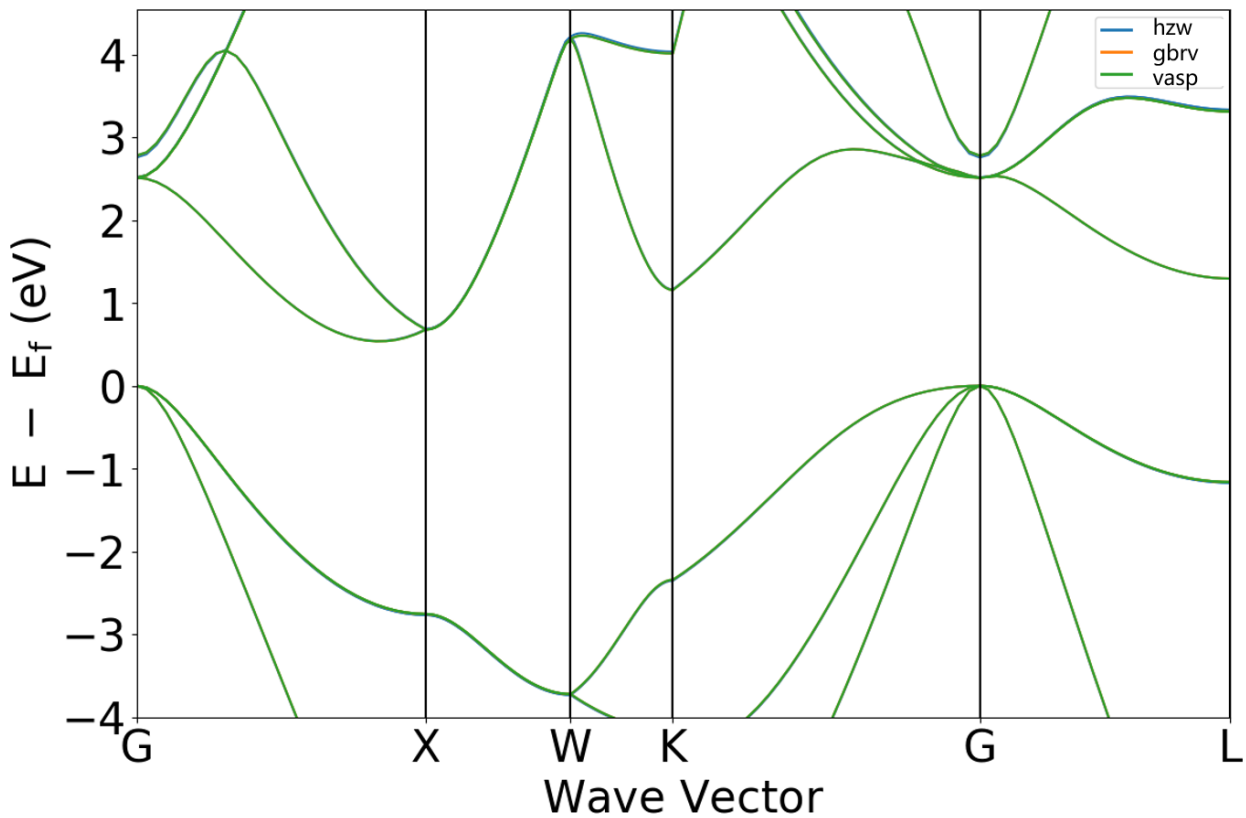
## 4.3 GBRV Pseudopotential

DS-PAW provides an interface for using external gbrv-formatted pseudopotentials (.pawpsp), with the corresponding parameter `sys.pseudoType` set to **11**. In this case, DS-PAW will read the pseudopotential files from the default path **./**. The gbrv pseudopotential library is a freely available set of pseudopotentials, offering files for a total of 64 elements. The download website is http://www.physics.rutgers.edu/gbrv/. When downloading, please note that DS-PAW supports the PAW format for Abinit. When using these pseudopotentials, the user needs to rename the pseudopotential files accordingly. For example, if using the LDA pseudopotential for silicon, the corresponding pseudopotential file should be renamed to Si_LDA.pawpsp and placed in the specified directory (which can be set via sys.pseudoPath).

## 4.4 Compare Pseudopotentials

### 4.4.1 $Si$ band structure calculation

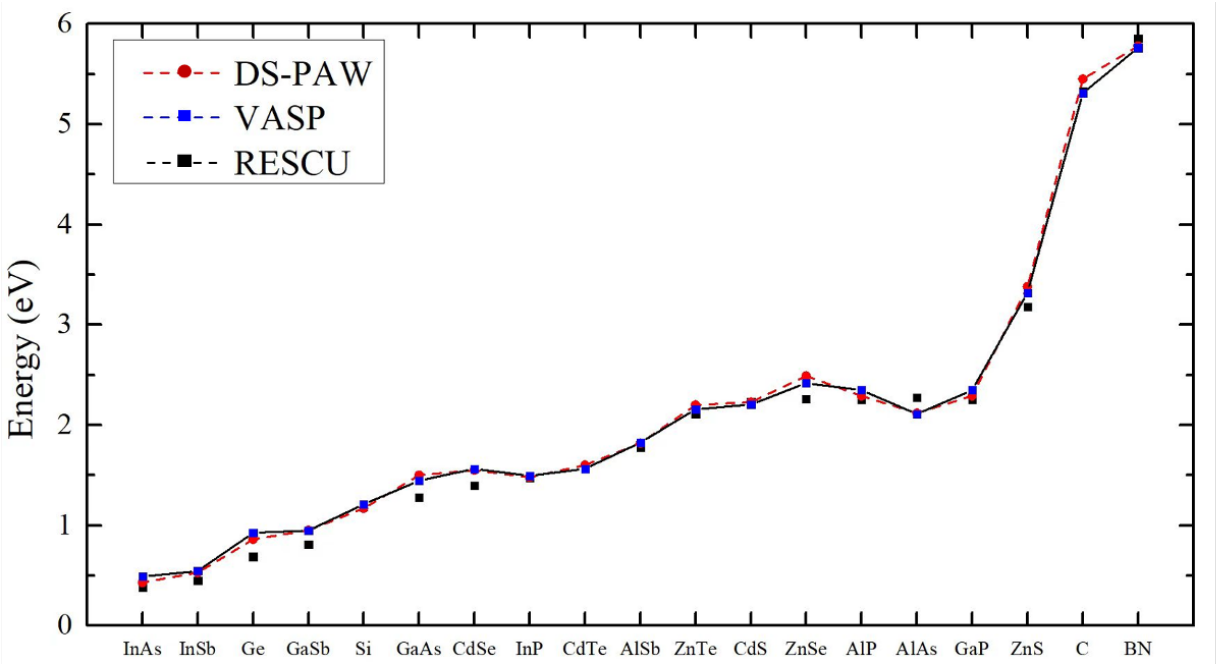
To compare the computational results of the three pseudopotentials, this section uses silicon (Si) as an example. Band structure calculations were performed using all three pseudopotentials. The figure below shows a comparison of the band structures. This comparison demonstrates a high degree of consistency among the three pseudopotentials in describing the Si band structure, thus validating the accuracy of the hzw pseudopotential.




***Data Source: The data for this figure was provided by a collaborator of Hongzhiwei.**

### 4.4.2 Multi-system band gap calculation

This section presents band gap calculations using HSE06 with version 1.0 pseudopotentials for multiple systems. The band gap values obtained are compared with those calculated using **VASP** and **RESCU** software, as reported in the literature, along with those from **DS-PAW** calculations. The resulting figure (shown below) further validates the accuracy of the hzw pseudopotentials.




***Data Source**: https://doi.org/10.1103/PhysRevB.97.075139

# 5

# Parameters Explanation

## 5.1 parameter parameter list

- *task*

---

- *sys.pseudoType*
- *sys.pseudoPath*
- *sys.structure*
- *sys.symmetry*
- *sys.symmetryAccuracy*
- *sys.functional*
- *sys.spin*
- *sys.spinDiff*
- *sys.soi*
- *sys.electron*
- *sys.hybrid*
- *sys.hybridType*
- *sys.hybridAlpha*
- *sys.hybridOmega*
- *sys.sol*
- *sys.solEpsilon*
- *sys.solTAU*
- *sys.solLambdaD*

- *sys.fixedP*
- *sys.fixedPConvergence*
- *sys.fixedPPotential*
- *sys.fixedPType*
- *sys.fixedPMaxIter*

---

- *cal.iniCharge*
- *cal.iniWave*
- *cal.cutoffFactor*
- *cal.cutoff*
- *cal.methods*
- *cal.smearing*
- *cal.sigma*
- *cal.kpoints*
- *cal.ksamping*
- *cal.totalBands*
- *cal.opticalGrid*
- *cal.iniFixedP*
- *cal.FFTGrid*
- *cal.supGrid*

---

- *io.charge*
- *io.elf*
- *io.potential*
- *io.wave*
- *io.band*
- *io.dos*
- *io.optical*
- *io.bader*
- *io.polarization*
- *io.magProject*
- *io.boundCharge*
- *io.outJsonFile*

---

- *scf.max*
- *scf.min*

- *scf.mixBeta*
- *scf.mixType*
- *scf.convergence*
- *scf.timeStep*

---

- *relax.max*
- *relax.freedom*
- *relax.methods*
- *relax.convergenceType*
- *relax.convergence*
- *relax.stepRange*
- *relax.pressure*

---

- *dos.range*
- *dos.resolution*
- *dos.project*

---

- *band.kpointsLabel*
- *band.kpointsCoord*
- *band.kpointsNumber*
- *band.project*
- *band.unfolding*
- *band.primitiveUVW*
- *band.EfShift*

---

- *optical.grid*
- *optical.KKEta*
- *optical.smearing*
- *optical.sigma*
- *optical.Emax*

---

- *potential.type*

---

- *corr.chargedSystem*
- *corr.dipol*

- *corr.dipolDirection*
- *corr.dftu*
- *corr.dftuForm*
- *corr.dftuElements*
- *corr.dftuOrbital*
- *corr.dftuU*
- *corr.dftuJ*
- *corr.VDW*
- *corr.VDWType*
- *corr.dipolEfield*
- *corr.dipolPosition*
- *corr.coreEnergy*

---

- *pcharge.bandIndex*
- *pcharge.kpointsIndex*
- *pcharge.sumK*

---

- *neb.springK*
- *neb.images*
- *neb.iniFin*
- *neb.method*
- *neb.convergenceType*
- *neb.convergence*
- *neb.stepRange*
- *neb.max*
- *neb.freedom*

---

- *frequency.dispOrder*
- *frequency.dispRange*

---

- *phonon.structureSize*
- *phonon.method*
- *phonon.type*
- *phonon.isDisplacement*
- *phonon.fdDisplacement*
- *phonon.iniPhonon*

- *phonon.qsamping*
- *phonon.qpoints*
- *phonon.qpointsLabel*
- *phonon.qpointsCoord*
- *phonon.qpointsNumber*
- *phonon.primitiveUVW*
- *phonon.dosRange*
- *phonon.dosResolution*
- *phonon.dosSigma*
- *phonon.dfptEpsilon*
- *phonon.nac*
- *phonon.thermal*
- *phonon.thermalRange*
- *phonon.eigenVectors*

---

- *elastic.dispOrder*
- *elastic.dispRange*

---

- *aimd.ensemble*
- *aimd.thermostat*
- *aimd.andersenProb*
- *aimd.noseMass*
- *aimd.latticeFCoeff*
- *aimd.atomFCoeffElements*
- *aimd.atomFCoeffs*
- *aimd.latticeMass*
- *aimd.pressure*
- *aimd.iniTemp*
- *aimd.finTemp*
- *aimd.timeStep*
- *aimd.totalSteps*

---

- *wannier.functions*
- *wannier.wannMaxIter*
- *wannier.disMaxIter*
- *wannier.disWin*

- *wannier.disFrozWin*
- *wannier.disEfShift*
- *wannier.interpolatedBand*
- *wannier.kpointsLabel*
- *wannier.kpointsCoord*
- *wannier.kpointsNumber*
- *wannier.kmeshTolerance*
- *wannier.outStep*
- *WannProj*

---

## 5.2 Detail parameter description

**Parameter Name:** *task*

**Default:** None

**Optional values:** `scf/relax/dos/band/optical/potential/elf/pcharge/neb/frequency/phonon/elastic/aimd/epsilon/wannier`

**Description:** The *task* parameter specifies the calculation type and is mandatory. `scf/relax` can be a from-scratch calculation (without setting `cal.iniCharge` and `cal.iniWave`) or import charge density or wave functions (by setting `cal.iniCharge` and `cal.iniWave`). `dos/band/optical/potential/elf` are post-processing calculations that require reading charge density. When importing charge density, you can optionally import the wave function ( **cal.iniCharge** must be set, **cal.iniWave** is optional);

**Case:** task = scf

---

**Parameter Name:** *sys.pseudoType*

**Default:** -1

**Optional Values:** -1/10/11

**Description:** The `sys.pseudoType` parameter sets the pseudopotential format required for **DS-PAW** calculations; -1 indicates the use of hzw pseudopotentials (*.paw). Currently, DS-PAW supports hzw pseudopotentials for 72 elements: **H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I Xe Cs Ba La Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn**.*

**Description:** 10 represents external potcar format pseudopotentials (*.potcar), and 11 represents external pawpsp format pseudopotentials (*.pawpsp).

**Example:** sys.pseudoType = -1

---

**Parameter Name:** *sys.pseudoPath*

**Default:** When `sys.pseudoType` = -1, this parameter does not need to be set, and the program can only read pseudopotential files from the installation path **/pseudopotential**; `sys.pseudoType` = 10, the default value is **./**; `sys.pseudoType` = 11, the default value is **./**;

**Description:** The `sys.pseudoPath` parameter sets the path where the pseudopotentials required for **DS-PAW** calculations are located; it generally does not need to be set manually, as it reads from the default storage path when reading hzw pseudopotentials and defaults to the current path when reading external pseudopotentials.

**Example:** sys.pseudoPath = ./

---

**Parameter Name:** *sys.structure*

**Default:** atoms.as

**Optional Values:** .as / .h5 / .json

**Description:** The `sys.structure` parameter sets the path to the structure file, supporting `.as`, `.h5`, and `.json` formats, with both absolute and relative paths allowed; DS-PAW generates the `relax.h5` file by default after structural relaxation, so you can directly set `sys.structure = relax.h5`. Read the relaxed structure for calculation; (.json files are currently supported but not recommended, DS-PAW will completely eliminate the JSON format output in iterative versions.)

**Example:** sys.structure = relax.h5

---

**Parameter Name:** *sys.symmetry*

**Default value:** true

**Optional Values:** true/false

**Description:** The parameter `sys.symmetry` indicates whether symmetry analysis is performed during DS-PAW calculations;

**Example:** sys.symmetry = false

---

**Parameter Name:** *sys.symmetryAccuracy*

**Default value:** 1.0e-5

**Allowed values:** real

**Description:** The *sys.symmetryAccuracy* parameter specifies the accuracy of the symmetry analysis during DS-PAW calculations;

**Example:** sys.symmetryAccuracy = 1.0e-6

---

**Parameter Name:** *sys.functional*

**Default value:** LDA

**Options:** LDA/PBE/REVPBE/RPBE/PBESOL/vdw-optPBE/vdw-optB88/vdw-optB86b/vdw-DF/vdw-DF2/vdw-revDF2

**Description:** The `sys.functional` parameter specifies the functional type for DS-PAW. If **sys.functional=LDA**, the LDA pseudopotentials in the specified path will be read; pseudopotentials starting with vdw correspond to van der Waals correction methods for the functional.

**Example:** sys.functional = PBESOL

---

**Parameter Name:** *sys.spin*

**Default value:** none

**Options:** none/collinear/non-collinear

**Description:** The `sys.spin` parameter specifies the spin properties to be calculated; **none** indicates no spin, **collinear** indicates collinear spin, and **non-collinear** indicates general spin;

**Example:** sys.spin = collinear

---

**Parameter Name:** *sys.spinDiff*

**Default value:** None

**Optional Values:** $[0, \infty)$

**Description:** Sets the difference in the number of up and down spin electrons;

**Example:** sys.spinDiff = 1

---

**Parameter Name:** *sys.soi*

**Default:** false

**Possible values:** true/false

**Description:** `sys.soi` indicates whether to consider spin-orbit coupling; spin-orbit coupling only takes effect when sys.spin=non-collinear;

**Example:** sys.soi = true

---

**Parameter Name:** *sys.electron*

**Default:** The sum of all valence electrons

**Optional values:** real

**Description:** The `sys.electron` parameter specifies the total number of valence electrons; DS-PAW calculates charged systems by introducing a background charge.

**Case:** sys.electron = 12

---

**Parameter Name:** *sys.hybrid*

**Default:** false

**Allowed values:** true/false

**Description:** The `sys.hybrid` parameter specifies whether to use a hybrid functional. *true* indicates the introduction of a hybrid functional, while *false* indicates its absence. `sys.hybrid` is only effective when *task* = *scf* or *relax*. When `sys.hybrid` is set to *true*, `sys.functional` is no longer effective.

**Example:** sys.hybrid = true

---

**Parameter Name:** *sys.hybridType*

**Default value:** HSE06

**Possible values:** PBE0/HSE03/HSE06/B3LYP

**Description:** The `sys.hybridType` parameter specifies the type of hybrid functional; this parameter only takes effect when sys.hybrid = true;

**Example:** sys.hybridType = HSE06

---

**Parameter Name:** *sys.hybridAlpha*

**Default:** When `sys.hybridType` = PBE0, the default value is **0.25**, when `sys.hybridType` = HSE06, the default value is **0.25**, and when `sys.hybridType` = HSE03, the default value is **0.25**.

**Possible values:** real

**Description:** The `sys.hybridAlpha` parameter specifies the coefficient of the exact exchange correlation functional in the hybrid functional; this parameter is only effective when sys.hybrid = true;

**Example:** sys.hybridAlpha = 0.20

---

**Parameter Name:** *sys.hybridOmega*

**Default:** When `sys.hybridType` = PBE0, the default value is **0**, when `sys.hybridType` = HSE06, the default value is **0.2**, and when `sys.hybridType` = HSE03, the default value is **0.3**.

**Possible values:** real

**Description:** The `sys.hybridOmega` parameter specifies the screening coefficient for the hybrid functional; this parameter is only active when sys.hybrid = true;

**Example:** sys.hybridOmega = 0.2

---

**Parameter Name:** *sys.sol*

**Default:** false

**Allowed values:** false/true

**Description:** The `sys.sol` parameter specifies whether to apply the implicit solvation model;

**Example:** sys.sol = true

---

**Parameter Name:** *sys.solEpsilon*

**Default:** 78.4

**Optional values:** real

**Description:** The `sys.solEpsilon` parameter specifies the solvent dielectric constant, with a default value of the dielectric constant of water.

**Example:** sys.solEpsilon = 80

---

**Parameter Name:** *sys.solTAU*

**Default:** 5.25E-4

**Possible values:** real

**Description:** The `sys.solTAU` parameter specifies the magnitude of the effective interfacial tension per unit area, in units of eV/Å^2. It is recommended that this parameter be set to a value less than 1e-3;

**Example:** sys.solTAU = 0

---

**Parameter Name:** *sys.solLambdaD*

**Default value:** None

**Possible values:** real

**Description:** The `sys.solLambdaD` parameter specifies the Debye length in the Poisson-Boltzmann equation, in Å. If not set, the Poisson equation is used, and the screening effect of the double-layer ions on the electrostatic potential is ignored.

**Example:** sys.solLambdaD = 3.04

**Note**

1. The Debye length, *sys.solLambdaD*, is calculated as $\lambda_D = \sqrt{\frac{\varepsilon\varepsilon_o k_B T}{2c^0 z^2 q^2}}$

   The Debye length for a 1M aqueous solution of monovalent cations and anions (+/-1 charge) is: 3.04 Å

---

**Parameter Name:** *sys.fixedP*

**Default value:** false

**Options:** false/true

**Description:** The *sys.fixedP* parameter is a switch to control the fixed potential calculation, currently only compatible with *task* = *scf*.

**Example:** sys.fixedP = true

---

**Parameter Name:** *sys.fixedPConvergence*

**Default value:** 0.01

**Allowed values:** real

**Description:** The `sys.fixedPConvergence` parameter specifies the convergence accuracy for fixed potential calculations. The calculation terminates when the difference (delta_electron) between two consecutive self-consistent calculations is less than the convergence accuracy.

**Example:** sys.fixedPConvergence = 0.01

---

**Parameter Name:** *sys.fixedPPotential*

**Default Value:** None

**Allowed values:** real

**Description:** The *sys.fixedPPotential* parameter specifies the target electrode potential value for the fixed potential calculation, with the default reference electrode potential being the Standard Hydrogen Electrode (SHE).

**Example:** sys.fixedPPotential = 5.4723

---

**Parameter Name:** *sys.fixedPType*

**Default value:** SHE

**Options:** SHE/PZC

**Description:** The `sys.fixedPType` parameter specifies the type of potential for the potential values given by `sys.fixedPPotential`. SHE uses the standard hydrogen electrode (SHE) potential as the reference value, while PZC uses the zero charge potential as the reference value;

**Example:** sys.fixedPType = SHE

---

**Parameter Name:** *sys.fixedPMaxIter*

**Default value:** 60

**Optional values:** int

**Description:** The `sys.fixedPMaxIter` parameter specifies the maximum number of iterations for fixed potential calculations.

**Example:** sys.fixedPMaxIter = 100

---

**Parameter Name:** *cal.iniCharge*

**Default Value:** None

**Optional values:** Path to the rho.bin file

**Description:** The *cal.iniCharge* parameter indicates the path to the **rho.bin** file obtained from a DS-PAW self-consistent or structural relaxation calculation, which can be specified for subsequent calculations; When *task=scf/relax*, if reading the previous charge density is not required, *cal.iniCharge* is not set, and if it is required to read the previous charge density, *cal.iniCharge* is set. When *task=dos/band/potential/elf*, *cal.iniCharge* must be set to specify the path to **rho.bin**. Both relative and absolute paths are supported.

**Example:** cal.iniCharge = ../scf/rho.bin

---

**Parameter Name:** *cal.iniWave*

**Default value:** None

**Allowed value:** Specify the path to wave.bin

**Description:** The `cal.iniWave` parameter indicates the path to the wave function file **wave.bin** obtained from DS-PAW self-consistent or structure relaxation calculations, which can be used for subsequent calculations; if this parameter is not set, it means that **wave.bin** will not be read; the file path supports both relative and absolute paths;

**Example:** cal.iniWave = ../scf/wave.bin

---

**Parameter Name:** *cal.cutoffFactor*

**Default value:** 1.0

**Allowed value:** real

**Description:** `cal.cutoffFactor` represents the coefficient for the cutoff energy parameter `cal.cutoff`. When **cal.cutoffFactor=1.5**, the cutoff energy used in DS-PAW calculations is `cal.cutoff*1.5`. The pseudopotentials in the DS-PAW2022A version have all been tested, and the default value of 1.0 for cutoffFactor satisfies most computational requirements;

**Example:** cal.cutoffFactor = 1.0

---

**Parameter Name:** *cal.cutoff*

**Default value:** The maximum cutoff energy used in the pseudopotential for the current calculation;

**Allowed value:** real

**Description:** The `cal.cutoff` parameter represents the cutoff energy of plane waves used in the calculation by the DS-PAW software. The built-in cutoff energy (ecutoff) for each pseudopotential file can be viewed in the **/pseudopotential** directory, such as reading the ecutoff of O_PBE as 480 eV from the O_PBE.paw file.

**Example:** cal.cutoff = 480

---

**Parameter Name:** *cal.methods*

**Default value:** 1 (When sys.hybrid = true, the default value is 4)

**Allowed value:** 1/2/3/4/5

**Description:** `cal.methods` indicates the method used for the self-consistent electronic part optimization, where 1 represents the `BD(block Davidson)` method and 2 represents the `RM(residual minimization)` method; 3 represents the combination of the `RM(residual minimization)` method and the `BD(block Davidson)` method; 4 represents the `damped MD` (damped molecular dynamics) method; 5 represents the `conjugated gradient` (conjugate gradient) method; among which 4 and 5 can be used with hybrid functionals;

**Example:** cal.methods = 1

---

**Parameter Name:** *cal.smearing*

**Default value:** 1

**Allowed value:** 1/2/3/4

**Description:** `cal.smearing` specifies the method used to set partial occupancies for each wave function Gaussian smearing/Fermi-smearing/Methfessel-Paxton order 1/tetrahedron method with Blochl corrections;

**Example:** cal.smearing = 2

---

**Parameter Name:** *cal.sigma*

**Default value:** 0.2

**Allowed value:** real

**Description:** `cal.sigma` represents the broadening when setting partial occupation numbers using finite temperature methods;

**Example:** cal.sigma = 0.01

---

**Parameter Name:** *cal.kpoints*

**Default value:** [1,1,1]

**Allowed value:** 3*1 int array

**Description:** `cal.kpoints` specifies the sampling size of the k-point grid in the Brillouin zone for DS-PAW settings;

**Example:** cal.kpoints = [9,9,9]

---

**Parameter Name:** *cal.ksamping*

**Default value:** MP

**Allowed value:** MP/G

**Description:** `cal.ksampling` indicates the method for automatically generating the k-point grid in the Brillouin zone by DS-PAW, `Monhkorst-Pack` method / `Gamma centered` method;

**Example:** cal.ksampling = G

---

**Parameter Name:** *cal.totalBands*

**Default value:** Related to the number of valence electrons in the system

**Optional values:** int

**Description:** `cal.totalBands` represents the total number of bands included in the DS-PAW calculation;

**Example:** cal.totalBands = 100

---

**Parameter Name:** *cal.opticalGrid*

**Default value:** 2000

**Allowed Values:** int

**Description:** *cal.opticalGrid* represents the number of grid points in the energy region when calculating optical properties in DS-PAW. It only takes effect when *io.optical* is enabled.

**Example:** cal.opticalGrid = 2000

---

**Parameter Name:** *cal.iniFixedP*

**Default value:** None

**Allowed value:** The path to the h5 file output by the constant potential calculation

**Description:** The `cal.iniFixedP` specifies the path to the h5 file from the previous constant potential calculation, which DS-PAW reads to perform a continuation of the constant potential calculation;

**Example:** cal.iniFixedP = ./scf.h5

---

**Parameter Name:** *cal.FFTGrid*

**Default value:** Depends on the parameters *cal.cutoff* and *cal.cutoffFactor*

**Allowed value:** 3*1 int array

**Description:** `cal.FFTGrid` specifies the number of grid points along three lattice directions for the FFT grid of the unit cell;

**Example:** cal.FFTGrid = [16,16,16]

---

**Parameter Name:** *cal.supGrid*

**Default value:** false

**Allowed value:** true/false

**Description:** The `cal.supGrid` is a switch to enable or disable the use of support FFTGrid, which can increase the density of the FFT-Grid;

**Example:** cal.supGrid = true

---

**Parameter Name:** *io.charge*

**Default value:** true

**Allowed value:** true/false

**Description:** Controls whether to output the charge density files `rho.bin` and `rho.h5`; when io.charge=true, the `rho.bin` and `rho.h5` files are generated;

**Example:** io.charge = true

---

**Parameter Name:** *io.elf*

**Default value:** false

**Allowed value:** false/true

**Description:** Output ELF data results; this parameter takes effect when task=scf/relax; does not support setting sys.spin=non-collinear simultaneously

**Example:** io.elf = true

---

**Parameter Name:** *io.potential*

**Default value:** false

**Allowed value:** false/true

**Description:** Output data results of the potential function; this parameter is effective when task=SCF/relax; when io.potential=true, you can choose `potential.type` to set the type of the output potential function;

**Example:** io.potential = true

---

**Parameter Name:** *io.wave*

**Default value:** true when task is wannier and wave.bin file is not read, false for other tasks

**Allowed value:** false/true

**Description:** Output the binary file of the wave function *wave.bin*; when *io.wave=true*, generate the *wave.bin* file;

**Example:** io.wave = true

---

**Parameter Name:** *io.band*

**Default value:** false

**Allowed value:** false/true

**Description:** Whether to directly calculate the band switching when task=scf; when io.band=true, all band calculation parameters take effect;

**Example:** io.band = true

---

**Parameter Name:** *io.dos*

**Default value:** false

**Allowed value:** false/true

**Description:** A switch to directly calculate the density of states when task=scf; when io.dos=true, all density of states calculation parameters take effect;

**Example:** io.dos = true

---

**Parameter Name:** *io.optical*

**Default value:** false

**Allowed value:** false/true

**Description:** Controls whether to perform optical property calculations; io.optical=true is only effective when task=scf is set, and when this parameter is active, the corresponding scf.h5 file will be written with optical property data;

**Example:** io.optical = true

---

**Parameter Name:** *io.bader*

**Default value:** false

**Allowed value:** false/true

**Description:** Controls whether to perform Bader charge calculation; io.bader=true only takes effect when task=scf is set, and when this parameter is active, the corresponding scf.h5 file will be written with Bader charge data;

**Example:** io.bader = true

---

**Parameter Name:** *io.polarization*

**Default value:** false

**Allowed value:** false/true

**Description:** Controls whether to perform iron polarization calculation; io.polarization=true only takes effect when task=scf is set, and when this parameter is active, the corresponding scf.h5 file will be written with iron polarization data;

**Example:** io.polarization = true

---

**Parameter Name:** *io.magProject*

**Default value:** true when sys.spin=collinear or sys.spin=non-collinear, false otherwise

**Allowed value:** false/true

**Description:** In magnetic moment calculations, controls whether to write projected magnetic moment data to the corresponding h5 output file;

**Example:** io.magProject = true

---

**Parameter Name:** *io.boundCharge*

**Default value:** false

**Allowed value:** true/false

**Description:** Controls whether to output solvent bound charge density files when an implicit solvent model is introduced;

**Example:** io.boundCharge = true

---

**Parameter Name:** *io.outJsonFile*

**Default value:** true

**Allowed value:** true/false

**Description:** Controls whether to output a JSON-formatted output file;

**Example:** io.outJsonFile = false

---

**Parameter Name:** *scf.max*

**Default value:** 60

**Allowed value:** int

**Description:** `scf.max` specifies the maximum number of electronic steps in a DS-PAW self-consistent field calculation;

**Example:** scf.max = 100

---

**Parameter Name:** *scf.min*

**Default value:** 2

**Allowed value:** int

**Description:** `scf.min` represents the minimum number of electronic steps for self-consistent calculations in DS-PAW;

**Example:** scf.min = 5

---

**Parameter Name:** *scf.mixBeta*

**Default value:** 0.4

**Allowed value:** real

**Description:** `scf.mixBeta` represents the Beta value of the electronic mixing algorithm used in DS-PAW self-consistent calculations;

**Example:** scf.mixBeta = 0.2

---

**Parameter Name:** *scf.mixType*

**Default value:** Pulay

**Allowed value:** Broyden/Kerker/Pulay

**Description:** `scf.mixType` specifies the type of electronic mixing algorithm used in DS-PAW self-consistent calculations, currently supporting the **Broyden method**, **Kerker method**, and **Pulay method**;

**Example:** scf.mixType = Pulay

---

**Parameter Name:** *scf.convergence*

**Default value:** 1.0e-4

**Allowed value:** real

**Description:** `scf.convergence` specifies the energy convergence criterion for the DS-PAW self-consistent calculation;

**Example:** scf.convergence = 1.0e-5

---

**Parameter Name:** *scf.timeStep*

**Default value:** 0.4

**Allowed value:** real

**Description:** The parameter `scf.timeStep` controls the step size when cal.methods=4/5; When cal.methods = 4, scf.timeStep determines the MD step size; a too small step size will increase the number of steps required for convergence, while a too large step size may cause the scf calculation to diverge. When cal.methods = 5, scf.timeStep determines the initial step size; a too large step size may cause the scf calculation to become unstable, while a too small step size may result in insufficient accuracy.

**Example:** scf.timeStep = 0.4

---

**Parameter Name:** *relax.max*

**Default value:** 60

**Allowed value:** int

**Description:** `relax.max` represents the maximum number of ion steps during the relaxation of the DS-PAW structure;

**Example:** relax.max = 300

---

**Parameter Name:** *relax.freedom*

**Default value:** atom

**Allowed value:** atom/volume/all/atom&shape

**Description:** `relax.freedom` specifies the degrees of freedom for the relaxation of the DS-PAW structure; atom indicates relaxation of only atomic positions; volume indicates relaxation of only the lattice volume; all indicates relaxation of atomic positions, lattice volume, and unit cell shape; atom&shape indicates relaxation of atomic positions and lattice shape;

**Example:** relax.freedom = atom

---

**Parameter Name:** *relax.methods*

**Default value:** CG

**Allowed value:** CG/DMD/QN

**Description:** `relax.methods` specifies the relaxation method for the DS-PAW structure, where CG stands for Conjugate Gradient method; DMD for Damped Molecular Dynamics method; QN for Quasi-Newton method;

**Example:** relax.methods = CG

---

**Parameter Name:** *relax.convergenceType*

**Default value:** force

**Allowed value:** force/energy

**Description:** The `relax.convergenceType` specifies the choice of convergence criterion in the relaxation calculation, with options being force or energy as the convergence standard;

**Example:** relax.convergenceType = energy

---

**Parameter Name:** *relax.convergence*

**Default value:** 0.05/1e-4

**Allowed value:** real

**Description:** `relax.convergence` specifies the convergence criterion for atomic forces or energy during the relaxation of a DS-PAW structure; the default value is 0.05 when forces are used as the convergence standard, and 1e-4 when energy is used as the convergence standard;

**Example:** relax.convergence = 0.01

---

**Parameter Name:** *relax.stepRange*

**Default value:** 0.5

**Allowed value:** real

**Description:** `relax.stepRange` represents the scaling constant within the structural relaxation;

**Example:** relax.stepRange = 0.2

---

**Parameter Name:** *relax.pressure*

**Default value:** 0

**Allowed value:** real

**Description:** `relax.pressure` indicates that the structure optimization will be performed under a specific external pressure, and can also be used to correct Pullay stress error, unit kbar ;

**Example:** relax.pressure = 100

---

**Parameter Name:** *dos.range*

**Default value:** [-10,10]

**Allowed value:** 2*1 array

**Description:** `dos.range` indicates the energy interval for density of states calculation when task=dos;

**Example:** dos.range = [-15,15]

---

**Parameter Name:** *dos.resolution*

**Default value:** 0.05

**Allowed value:** real

**Description:** `dos.resolution` indicates the energy interval accuracy for density of states calculation when task=dos;

**Example:** dos.resolution = 0.1

---

**Parameter Name:** *dos.project*

**Default value:** false

**Allowed value:** false/true

**Description:** The *dos.project* parameter controls the projected density of states; when *task=dos*, *dos.project* is **false/true**; if projection is enabled, *dos.project = true*, and the projected density of states information will be saved in the *dos.h5* file; if projection is not enabled, *dos.project = false*;

**Example:** dos.project = true

---

**Parameter Name:** *band.kpointsLabel*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** This parameter is only effective when task=band; `band.kpointsLabel` is the high-symmetry point labels for band calculation, the size of the `band.kpointsLabel` array is **1/3** of the size of the `band.kpointsCoord` array; larger by **1** than the size of the `band.kpointsNumber` array;

**Example:** band.kpointsLabel = [G,M,K,G]

---

**Parameter Name:** *band.kpointsCoord*

**Default value:** None

**Allowed value:** 3n*1 real array

**Description:** This parameter is only effective when task=band; **band.kpointsCoord** represents the fractional coordinates of high-symmetry points during band calculation, and the data size of **band.kpointsCoord** is **3 times** the data size of **band.kpointsLabel**;

**Example:** band.kpointsCoord = [0, 0, 0, 0.5, 0.5, 0.5, 0, 0, 0.5, 0, 0, 0]

---

**Parameter Name:** *band.kpointsNumber*

**Default value:** None

**Allowed value:** (n-1)*1 int array/ 1*1 int array

**Description:** This parameter is only effective during band calculations; **band.kpointsNumber** is the number of K points between each pair of adjacent high-symmetry points

- – When the parameter length is (n-1)*1 int array, **band.kpointsNumber** is one less in size than the data size of **band.kpointsLabel**
- – When the parameter length is a 1*1 int array, it performs equal-density point distribution for all high-symmetry points based on the given parameter; the final number of equally-distributed points can be read from **band.kpointsNumber** in DS-PAW.log;

**Example:** band.kpointsNumber = [100]

---

**Parameter Name:** *band.project*

**Default value:** false

**Allowed value:** false/true

**Description:** The `band.project` parameter controls the projection of bands; when task= band, if band.project is set to true, the projection band information will be saved in the `band.h5` file; if projection is not enabled, set band.project = false;

**Example:** band.project = true

---

**Parameter Name:** *band.unfolding*

**Default value:** false

**Allowed value:** false/true

**Description:** The `band.unfolding` parameter is a switch for band unfolding; when task= band, `band.unfolding` takes effect (io.band = true does not take effect), and if `band.unfolding` is set to true, the unfolded band data will be saved in the `band.h5` file;

**Example:** task = band, band.unfolding = true

---

**Parameter Name:** *band.primitiveUVW*

**Default value:** None

**Allowed value:** 9*1 real array

**Description:** The `band.primitiveUVW` ensures that when performing folding calculations, the product of the lattice constants of the supercell multiplied by the UVW coefficients equals the lattice vectors of the primitive cell;

**Example:** band.primitiveUVW = [0.0, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.0]

---

**Parameter Name:** *band.EfShift*

**Default value:** true when task=band, false for other tasks

**Allowed value:** true/false

**Description:** The `band.EfShift` parameter indicates whether to read EFermi from rho.bin when task=band, and it takes effect only when task=band;

**Example:** band.EfShift = true

---

**Parameter Name:** *optical.grid*

**Default value:** 2000

**Allowed value:** int

**Description:** `optical.grid` indicates the number of grid points in the energy region when calculating optical properties with DS-PAW, and takes effect only when io.optical and task=optical are specified;

**Example:** optical.grid = 2000

---

**Parameter Name:** *optical.KKEta*

**Default value:** EnergyAxe resolution*0.99

**Allowed value:** real

**Description:** `optical.KKEta` is the $\eta$ value used when solving the real part of the dielectric function using the Kramers-Kroning relationship. Using the default value may result in very rough results. Increasing $\eta$ can make the results smoother, but it may introduce some errors in the calculation of the dielectric function values in the low-frequency region. It is not recommended to use excessively large $\eta$ values; instead, it is suggested to increase the number of grid points (optical.grid) to achieve smoother results. (In older versions without this parameter, the $\eta$ value was 0.1)

**Example:** optical.KKEta = 0.1

---

**Parameter Name:** *optical.smearing*

**Default value:** 1

**Allowed value:** 1/2/3

**Description:** `optical.smearing` determines the smearing algorithm for energy broadening during optical calculations. 1/2/3 correspond to Gaussian smearing/Fermi smearing/Methfessel-Paxton order 1;

**Example:** optical.smearing = 1

---

**Parameter Name:** *optical.sigma*

**Default value:** 0.05

**Allowed value:** real

**Description:** `optical.sigma` determines the width of the broadening when using the expansion algorithm determined by optical.smearing;

**Example:** optical.sigma = 0.05

---

**Parameter Name:** *optical.Emax*

**Default value:** Maximum energy of the unoccupied state*1.2 (eV)

**Allowed value:** real

**Description:** `optical.Emax` determines the maximum value of frequency (EnergyAxe) during optical calculations;

**Example:** optical.Emax = 20

---

**Parameter Name:** *potential.type*

**Default value:** total

**Allowed value:** total/hartree/all

**Description:** `potential.type` controls the output type of the electrostatic potential; when potential.type = hartree, the `potental.h5` file writes the electrostatic potential (sum of ionic potential and Hartree potential), when potential.type = total, the `potental.h5` file writes the local potential (sum of electrostatic potential and exchange-correlation potential) data, when potential.type = all, the `potental.h5` file writes both types of potential;

**Example:** potential.type = all

---

**Parameter Name:** *corr.chargedSystem*

**Default value:** false

**Allowed value:** false/true

**Description:** `corr.chargedSystem` indicates whether the energy of charged block systems can be corrected when calculating charged systems;

**Example:** corr.chargedSystem = true

---

**Parameter Name:** *corr.dipol*

**Default value:** false

**Allowed value:** false/true

**Description:** `corr.dipol` indicates the introduction of an artificial potential field (dipole correction) to address the issue of uneven vacuum potential;

**Example:** corr.dipol = true

---

**Parameter Name:** *corr.dipolDirection*

**Default value:** None

**Allowed value:** a/b/c/all

**Description:** `corr.dipolDirection` indicates the direction of the dipole correction, where a/b/c represent the directions of the three lattice constants, and all indicates all directions, applicable for isolated molecule calculations;

**Example:** corr.dipolDirection = c

---

**Parameter Name:** *corr.dipolPosition*

**Default value:** None

**Allowed value:** 3*1 real array

**Description:** `corr.dipolPosition` represents the relative position of the dipole in the unit cell;

**Example:** corr.dipolPosition = [0.5, 0.5, 0.5]

---

**Parameter Name:** *corr.dipolEfield*

**Default value:** 0

**Allowed value:** real

**Description:** `corr.dipolEfield` represents the magnitude of the external electric field, in units of eV/Å, and this parameter is only effective when `corr.dipol = true` and `corr.dipolDirection` is set;

**Example:** corr.dipolEfield = 0.05

---

**Parameter Name:** *corr.dftu*

**Default value:** false

**Allowed value:** false/true

**Description:** `corr.dftu` indicates whether to introduce Hubbard U to handle strongly correlated systems;

**Example:** corr.dftu = true

---

**Parameter Name:** *corr.dftuForm*

**Default value:** 2

**Allowed value:** 1/2

**Description:** `corr.dftuForm` indicates which DFT+U method to select. 1 corresponds to the DFT+U+J method (Liechtensteins formulation), 2 corresponds to the DFT+U method (Dudarevs formulation);

**Example:** corr.dftuForm = 2

---

**Parameter Name:** *corr.dftuElements*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `corr.dftuElements` indicates the elements that require the addition of U;

**Example:** corr.dftuElements = [Ni,O]

---

**Parameter Name:** *corr.dftuOrbital*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `corr.dftuOrbital` indicates the orbitals that need to be added U on the selected elements;

**Example:** corr.dftuOrbital = [d,s]

---

**Parameter Name:** *corr.dftuU*

**Default value:** None

**Allowed value:** n*1 real array

**Description:** `corr.dftuU` indicates the size of the U value to be added to the selected orbit on the selected element;

**Example:** corr.dftuU = [8,1]

---

**Parameter Name:** *corr.dftuJ*

**Default value:** None

**Allowed value:** n*1 real array

**Description:** `corr.dftuJ` indicates the size of the J value to be added to the selected orbit on the selected element;

**Example:** corr.dftuJ = [0.95,0]

---

**Parameter Name:** *corr.VDW*

**Default value:** false

**Allowed value:** false/true

**Description:** `corr.VDW` indicates whether to introduce van der Waals corrections;

**Example:** corr.VDW = true

---

**Parameter Name:** *corr.VDWType*

**Default value:** D2G

**Allowed value:** D2G/D3G/D3BJ

**Description:** `corr.VDWType` indicates which van der Waals correction is used, D2G represents DFT-D2 of Grimmes method; D3G represents DFT-D3 of Grimmes method; D3BJ represents DFT-D3 with Becke-Jonson damping method;

**Example:** corr.VDWType = D3G

---

**Parameter Name:** *corr.coreEnergy*

**Default value:** false

**Allowed value:** true/false

**Description:** `corr.coreEnergy` indicates whether to use the initial state approximation to calculate the core electron energy levels;

**Example:** corr.coreEnergy = true

---

**Parameter Name:** *pcharge.bandIndex*

**Default value:** None

**Allowed value:** n*1 int array

**Description:** `pcharge.bandIndex` indicates the indices of bands used in the partial charge density calculation;

**Example:** pcharge.bandIndex = [1,3,4]

---

**Parameter Name:** *pcharge.kpointsIndex*

**Default value:** None

**Allowed value:** n*1 int array

**Description:** `pcharge.kpointsIndex` represents the indices of K points during partial charge density calculation;

**Example:** pcharge.kpointsIndex = [12,14]

---

**Parameter Name:** *pcharge.sumK*

**Default value:** false

**Allowed value:** false/true

**Description:** `pcharge.sumK` indicates whether to sum data of all K points and different bands after calculating the partial charge density and save the data.

**Example:** pcharge.sumK = true

---

**Parameter Name:** *neb.springK*

**Default value:** 5

**Allowed value:** real

**Description:** `neb.springK` represents the spring constant K in transition state calculations;

**Example:** neb.springK = 7

---

**Parameter Name:** *neb.images*

**Default value:** None

**Allowed value:** int

**Description:** `neb.images` represents the number of intermediate structures in transition state calculations;

**Example:** neb.images = 5

---

**Parameter Name:** *neb.iniFin*

**Default value:** false

**Allowed value:** true/false

**Description:** `neb.iniFin` indicates whether the initial and final structures are subjected to self-consistent calculations during transition state calculations, where true means self-consistent calculations are performed;

**Example:** neb.iniFin = true

---

**Parameter Name:** *neb.method*

**Default value:** QN

**Allowed value:** LBFGS/CG/QM/QN/QM2/FIRE

**Description:** `neb.method` specifies the algorithm used in transition state calculations;

**Example:** neb.method = QN

---

**Parameter Name:** *neb.freedom*

**Default value:** atom

**Allowed value:** atom/all

**Description:** `neb.freedom` represents the degrees of freedom for relaxation in transition state calculations, where you can choose to relax only atoms or allow the unit cell to be relaxed;

**Example:** neb.freedom = all

---

**Parameter Name:** *neb.convergenceType*

**Default value:** force

**Allowed value:** force/energy

**Description:** The `neb.convergenceType` specifies the convergence criterion in transition state calculations, where only force can be used as the convergence criterion when using LBFGS/CG/QM2/FIRE methods;

**Example:** neb.convergenceType = energy

---

**Parameter Name:** *neb.convergence*

**Default value:** 0.1/1e-4

**Allowed value:** real

**Description:** `neb.convergence` specifies the convergence criterion for forces or energies in transition state calculations; the default value is 0.1 when force is chosen as the convergence criterion, and 1e-4 when energy is chosen as the convergence criterion;

**Example:** neb.convergence = 0.01

---

**Parameter Name:** *neb.stepRange*

**Default value:** 0.1

**Allowed value:** real

**Description:** `neb.stepRange` indicates the step size for structural relaxation during transition state calculations;

**Example:** neb.stepRange = 0.01

---

**Parameter Name:** *neb.max*

**Default value:** 60

**Allowed value:** int

**Description:** `neb.max` specifies the maximum number of steps for structure relaxation in transition state calculations;

**Example:** neb.max = 300

---

**Parameter Name:** *frequency.dispOrder*

**Default value:** 1

**Allowed value:** 1/2

**Description:** `frequency.dispOrder` indicates the method of atomic vibration during frequency calculation, where 1 corresponds to the central difference method with two vibration modes, and 2 corresponds to four vibration modes;

**Example:** frequency.dispOrder = 2

---

**Parameter Name:** *frequency.dispRange*

**Default value:** 0.01

**Allowed value:** real

**Description:** `frequency.dispRange` represents the atomic displacement during frequency calculation;

**Example:** frequency.dispRange = 0.05

---

**Parameter Name:** *phonon.structureSize*

**Default value:** [1,1,1]

**Allowed value:** 3*1 int array

**Description:** `phonon.structureSize` indicates the size of the supercell used in the phonon calculation;

**Example:** phonon.structureSize = [2,2,2]

---

**Parameter Name:** *phonon.method*

**Default value:** fd

**Allowed value:** fd/dfpt

**Description:** The `phonon.method` specifies the method for phonon calculations; fd refers to the finite displacement method; dfpt refers to the density functional perturbation theory method;

**Example:** phonon.method = dfpt

---

**Parameter Name:** *phonon.type*

**Default value:** phonon

**Allowed value:** phonon/band/dos/bandDos

**Description:** `phonon.type` specifies which properties of phonons are calculated: phonon corresponds to calculating the force constant matrix or force set; band corresponds to calculating phonon bands; dos corresponds to calculating phonon density of states; bandDos corresponds to calculating both phonon bands and phonon density of states;

**Example:** phonon.type = bandDos

---

**Parameter Name:** *phonon.isDisplacement*

**Default value:** true

**Allowed value:** true/false

**Description:** `phonon.isDisplacement` indicates whether the displacement is calculated during the phonon calculation using the fd method;

**Example:** phonon.isDisplacement = true

---

**Parameter Name:** *phonon.fdDisplacement*

**Default value:** 0.01

**Allowed value:** real

**Description:** `phonon.fdDisplacement` represents the magnitude of displacement used in the phonon calculation by the fd (finite difference) method;

**Example:** phonon.fdDisplacement = 0.05

---

**Parameter Name:** *phonon.iniPhonon*

**Default value:** None

**Allowed value:** Specify the path to phonon.h5

**Description:** `phonon.iniPhonon` specifies the path for reading the force constant matrix or force set during phonon band or density of states calculations;

**Example:** phonon.iniPhonon = ../phonon/phonon.h5

---

**Parameter Name:** *phonon.qsamping*

**Default value:** MP

**Allowed value:** MP/G

**Description:** `phonon.qsamping` specifies the q-point sampling method in the Brillouin zone for phonon calculations, either the `Monkhorst-Pack` method or the `Gamma centered` method;

**Example:** phonon.qsamping = G

---

**Parameter Name:** *phonon.qpoints*

**Default value:** [1,1,1]

**Allowed value:** 3*1 int array

**Description:** `phonon.qpoints` represents the sampling size of the Q-space grid during phonon calculations;

**Example:** phonon.qpoints = [9,9,9]

---

**Parameter Name:** *phonon.qpointsLabel*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `phonon.qpointsLabel` indicates the labels of high-symmetry points during phonon band structure calculations;

**Example:** phonon.qpointsLabel = [G,M,K,G]

---

**Parameter Name:** *phonon.qpointsCoord*

**Default value:** None

**Allowed value:** 3n*1 real array

**Description:** `phonon.qpointsCoord` represents the coordinates of high-symmetry points during phonon band structure calculations;

**Example:** phonon.qpointsCoord = [0, 0, 0, 0.5, 0.5, 0.5, 0, 0, 0.5, 0, 0, 0]

---

**Parameter Name:** *phonon.qpointsNumber*

**Default value:** 51

**Allowed value:** int

**Description:** `phonon.qpointsNumber` represents the number of q-points between adjacent high-symmetry points in the phonon band;

**Example:** phonon.qpointsNumber = 100

---

**Parameter Name:** *phonon.primitiveUVW*

**Default value:** [1,0,0,0,1,0,0,0,1]

**Allowed value:** 9*1 real array

**Description:** For the phonon band calculation, the lattice vectors of the primitive cell are obtained by multiplying the lattice constants of the supercell by the UVW coefficients.

**Example:** phonon.primitiveUVW = [1,0,0,0,1,0,0,0,1]

---

**Parameter Name:** *phonon.dosRange*

**Default value:** [0, 40]

**Allowed value:** 2*1 real array

**Description:** `phonon.dosRange` indicates the energy range for the phonon density of states calculation;

**Example:** phonon.dosRange = [-15,15]

---

**Parameter Name:** *phonon.dosResolution*

**Default value:** 0.1

**Allowed value:** real

**Description:** `phonon.dosResolution` indicates the energy interval accuracy for the phonon density of states calculation;

**Example:** phonon.dosResolution = 0.01

---

**Parameter Name:** *phonon.dosSigma*

**Default value:** 0.1

**Allowed value:** real

**Description:** `phonon.dosSigma` represents the broadening used in the phonon density of states calculation;

**Example:** phonon.dosSigma = 0.1

---

**Parameter Name:** *phonon.dfptEpsilon*

**Default value:** false

**Allowed value:** false/true

**Description:** `phonon.dfptEpsilon` is a switch that controls the calculation of dielectric constant when phonon.method = dfpt;

**Example:** phonon.dfptEpsilon = true

---

**Parameter Name:** *phonon.nac*

**Default value:** true when phonon.dfptEpsilon = true

**Allowed value:** false/true

**Description:** When phonon.dfptEpsilon = true, if calculating band structure and density of states, phonon.nac is used as a switch for whether to use non-analytical term correction;

**Example:** phonon.nac = false

---

**Parameter Name:** *phonon.thermal*

**Default value:** false

**Allowed value:** false/true

**Description:** `phonon.thermal` is a switch that controls the calculation of thermodynamic properties when task=phonon and phonon.type=dos or phonon.type=bandDos;

**Example:** phonon.thermal = true

---

**Parameter Name:** *phonon.thermalRange*

**Default value:** [0,1200,10]

**Allowed value:** 3*1 real array

**Description:** `phonon.thermalRange` [min_T, max_T, $\delta$ T] specifies the temperature range for thermodynamic property calculations and the data storage interval;

**Example:** phonon.thermalRange = [0,1000,10]

---

**Parameter Name:** *phonon.eigenVectors*

**Default value:** false

**Allowed value:** false/true

**Description:** `phonon.eigenVectors` controls whether to output the eigenvectors of the dynamical matrix. When phonon.eigenVectors=true, EigenVectors output will be added under the BandInfo section in the phonon output file. EigenVectors>Size provides the size of the eigenvector matrix of the dynamical matrix (size: [NumberOfQPoints,

(NumberOfAtoms*3), NumberOfBand, (real, imag)]), EigenVectors>RowMajor indicates whether to output in row-major order, and EigenVectors>Values gives the values of the eigenvector matrix;

**Example:** phonon.eigenVectors = true

---

**Parameter Name:** *elastic.dispOrder*

**Default value:** 1

**Allowed value:** 1/2

**Description:** `elastic.dispOrder` indicates the method of atomic vibration during elastic constant calculation, where 1 corresponds to the central difference method (with two vibration modes), and 2 corresponds to the four vibration modes;

**Example:** elastic.dispOrder = 1

---

**Parameter Name:** *elastic.dispRange*

**Default value:** 0.01

**Allowed value:** real

**Description:** `elastic.dispRange` indicates the atomic displacement used in the calculation of elastic constants;

**Example:** elastic.dispRange = 0.05

---

**Parameter Name:** *aimd.ensemble*

**Default value:** NVE

**Allowed value:** NVE/NVT/NPT/NPH/SA

**Description:** `aimd.ensemble` indicates the ensemble used in molecular dynamics simulations; SA is an abbreviation for Simulated Annealing, corresponding to the simulation annealing process;

**Example:** aimd.ensemble = NVE

---

**Parameter Name:** *aimd.thermostat*

**Default value:** Depends on *aimd.ensemble*

**Allowed value:** andersen/noseHoover/langevin

**Description:** `aimd.thermostat` specifies the thermostat or barostat used in molecular dynamics simulations;

**Example:** aimd.thermostat = andersen

| Thermostat/Ensemble | NVE | NVT | NPT | NPH | SA |
|---|---|---|---|---|---|
| andersen | compatible* | compatible | incompatible | incompatible | incompatible |
| noseHoover | incompatible | compatible* | incompatible | incompatible | incompatible |
| langevin | incompatible | compatible | compatible* | compatible* | incompatible |

Note: * denotes default thermostat

---

**Parameter Name:** *aimd.andersenProb*

**Default value:** When *aimd.ensemble* is NVE, the default value is 0

**Allowed value:** When NVE, Allowed value is 0; When NVT, Allowed value is real (0 < x <= 1)

**Description:** The `aimd.andersenProb` controls the probability that atoms experience collisions under the Andersen thermostat;

**Example:** aimd.andersenProb = 0

---

**Parameter Name:** *aimd.noseMass*

**Default value:** 0

**Allowed value:** real (x >= 0)

**Description:** `aimd.noseMass` controls the effective mass of the Nose-Hoover thermostat;

**Example:** aimd.noseMass = 0

---

**Parameter Name:** *aimd.latticeFCoeff*

**Default value:** When *aimd.ensemble* is NPH, the default value is 0

**Allowed value:** 0 for NPH, real (x > 0) for NPT

**Description:** `aimd.latticeFCoeff` represents the magnitude of the lattice friction coefficient in the Langevin thermostat under NPT/NPH ensembles, with units of ps-1;

**Example:** aimd.latticeFCoeff = 10

---

**Parameter Name:** *aimd.atomFCoeffElements*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `aimd.atomFCoeffElements` represents the element names considered as Langevin atoms when using the Langevin thermostat. The naming convention is element name + underscore + custom field, such as Hf_1, and the element name in the structure.as file needs to be synchronized;

**Example:** aimd.atomFCoeffElements = [Hf_1,O_1]

---

**Parameter Name:** *aimd.atomFCoeffs*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `aimd.atomFCoeffs` represents the friction coefficients for Langevin atoms when using the Langevin thermostat, with units of ps-1. This value should correspond to the element names specified in `aimd.atomFCoeffElements`. For example, it assigns a value of 10 to the Hf_1 atom and a value of 5 to the O_1 atom;

**Example:** aimd.atomFCoeffElements = [Hf_1,O_1], aimd.atomFCoeffs = [10,5]

---

**Parameter Name:** *aimd.latticeMass*

**Default value:** 1000

**Allowed value:** real

**Description:** `aimd.latticeMass` represents the virtual mass of the cell degrees of freedom when using the Langevin barostat for NPT/NPH simulations, with units **amu**;

**Example:** aimd.latticeMass = 1000

---

**Parameter Name:** *aimd.pressure*

**Default value:** 0

**Allowed value:** real

**Description:** `aimd.pressure` represents the target pressure value of the system during NPT/NPH simulations, in units of **kbar**;

**Example:** aimd.pressure = 1000

---

**Parameter Name:** *aimd.iniTemp*

**Default value:** 0

**Allowed value:** real

**Description:** `aimd.iniTemp` represents the initial temperature during molecular dynamics simulation, in K;

**Example:** aimd.iniTemp = 1000

---

**Parameter Name:** *aimd.finTemp*

**Default value:** aimd.iniTemp

**Allowed value:** real

**Description:** `aimd.finTemp` represents the final temperature in the molecular dynamics simulation, this parameter is only effective when `aimd.ensemble = SA`; unit K;

**Example:** aimd.finTemp = 1000

---

**Parameter Name:** *aimd.timeStep*

**Default value:** 1

**Allowed value:** real

**Description:** `aimd.timeStep` represents the time step of the molecular dynamics simulation, in fs;

**Example:** aimd.timeStep = 1

---

**Parameter Name:** *aimd.totalSteps*

**Default value:** None

**Allowed value:** real

**Description:** `aimd.totalSteps` represents the total number of steps in the molecular dynamics simulation;

**Example:** aimd.totalSteps = 10000

---

**Parameter Name:** *wannier.functions*

**Default value:** None

**Allowed value:** int

**Description:** `wannier.functions` indicates the number of Wannier functions;

**Example:** wannier.functions = 8

---

**Parameter Name:** *wannier.wannMaxIter*

**Default value:** 200

**Allowed value:** int

**Description:** `wannier.wannMaxIter` represents the total number of iterations in the process of solving the maximally localized Wannier functions;

**Example:** wannier.wannMaxIter = 500

---

**Parameter Name:** *wannier.disMaxIter*

**Default value:** 100

**Allowed value:** int

**Description:** `wannier.disMaxIter` represents the maximum number of iterations for disentanglement;

**Example:** wannier.disMaxIter = 200

---

**Parameter Name:** *wannier.disWin*

**Default value:** [**lowest eigenvalue of the Hamiltonian obtained from self-consistent calculation**, **highest eigenvalue**]

**Allowed value:** 2*1 array

**Description:** `wannier.disWin` represents the disentanglement energy window, which defaults to including all bands;

**Example:** wannier.disWin = [-1000,1000]

---

**Parameter Name:** *wannier.disFrozWin*

**Default value:** None

**Allowed value:** 2*1 array

**Description:** `wannier.disFrozWin` represents the disentanglement window, where the states within this window remain unchanged during disentanglement;

**Example:** wannier.disFrozWin = [-10,10]

---

**Parameter Name:** *wannier.disEfShift*

**Default value:** false

**Allowed value:** true/false

**Description:** `wannier.disEfShift` indicates whether the energy input for wannier.disWin and wannier.disFrozWin is Ef=0;

**Example:** wannier.disEfShift = true

---

**Parameter Name:** *wannier.interpolatedBand*

**Default value:** false

**Allowed value:** true/false

**Description:** `wannier.interpolatedBand` indicates the switch for interpolating bands in the Wannier calculation;

**Example:** wannier.interpolatedBand = true

---

**Parameter Name:** *wannier.kpointsLabel*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `wannier.kpointsLabel` indicates the labels of high-symmetry points for interpolated band structures;

**Example:** wannier.kpointsLabel = [G,M,K,G]

---

**Parameter Name:** *wannier.kpointsCoord*

**Default value:** None

**Allowed value:** 3n*1 real array

**Description:** `wannier.kpointsCoord` indicates the fractional coordinates of the high-symmetry points for interpolated band structures;

**Example:** wannier.kpointsCoord = [0, 0, 0, 0.5, 0.5, 0.5, 0, 0, 0.5, 0, 0, 0]

---

**Parameter Name:** *wannier.kpointsNumber*

**Default value:** None

**Allowed value:** (n-1)*1 int array/ 1*1 int array

**Description:** This parameter is only effective when performing interpolated band calculations; **wannier.kpointsNumber** is the number of K points between adjacent high-symmetry points in the band.

- – When the parameter length is (n-1)*1 int array, **wannier.kpointsNumber** is one less than the data size of **wannier.kpointsNumber**
- – When the parameter length is a 1*1 int array, evenly distribute points around all high-symmetry points based on the given parameter; the final number of evenly distributed points can be read from **wannier.kpointsNumber** in DS-PAW.log;

**Example:** wannier.kpointsNumber = [100]

---

**Parameter Name:** *wannier.kmeshTolerance*

**Default value:** 1e-06

**Allowed value:** real

**Description:** `wannier.kmeshTolerance` determines whether two k-points are in the same shell;

**Example:** wannier.kmeshTolerance = 1e-06

---

**Parameter Name:** *wannier.outStep*

**Default value:** 20

**Allowed value:** int

**Description:** `wannier.outStep` specifies the interval at which wannier information is output when the task is set to wannier;

**Example:** wannier.outStep = 50

---

**Paramater Name:** *WannProj*

**Default value:** None

**Allowed value:** n*1 string array

**Description:** `WannProj` is the label defining the initial projection orbit in wannier calculations, used in `structure.as`;

**Example:**

```
1 Total number of atoms
2 2
3 Lattice
4 0.00 2.75 2.75
5 2.75 0.00 2.75
6 2.75 2.75 0.00
7 Direct WannProj
8 Si -0.125000000 -0.125000000 -0.125000000  [s,p]
9 Si 0.125000000 0.125000000 0.125000000     [s,p]
```

**Note**

1. The **WannProj** tag is set on line 7 of the structure.as file
2. The total number of projection orbits in this example is 2*(1+3) = 8

**Allowed value range:** DS-PAW supports **44** types of projection orbit names, divided into two categories, shown as follows:

**- First category**: Abbreviated names of orbits, corresponding to the total number of orbits for this type, with the two relationships shown in the table below:

| name | number of projections |
|---|---|
| [s] | 1 |
| [p] | 3 |
| [d] | 5 |
| [f] | 7 |
| [sp] | 2 |
| [sp2] | 3 |
| [sp3] | 4 |
| [sp3d] | 5 |
| [sp3d2] | 6 |

**- Second category**: The name of a specific orbit, with each array ([ ]) corresponding to **1** projection orbit:

| |
|---|
| [px] [py] [pz] |
| [dxy] [dyz] [dxz] [dz2] [dx2-y2] |
| [fz3] [fxz2] [fyz2] [fxyz] [fz(x2-y2)] [fx(x2-3y2)] [fy(3x2-y2)] |
| [sp-1] [sp-2] |
| [sp2-1] [sp2-2] [sp2-3] |
| [sp3-1] [sp3-2] [sp3-3] [sp3-4] |
| [sp3d-1] [sp3d-2] [sp3d-3] [sp3d-4] [sp3d-5] |
| [sp3d2-1] [sp3d2-2] [sp3d2-3] [sp3d2-4] [sp3d2-5] [sp3d2-6] |

**Note**

1. When the initial orbit is not defined (see *Quickstart* section 2.30), the program executes a randomly selected initial projection.

# 6

# Output File Format Specification

The DS-PAW 2023A versions default output files in **JSON** format can be directly analyzed and processed using Device Studio. Additionally, the output files now support the **hdf5** format. You can download vitables (run *pip install vitables* in a Python environment) or HDFView to view the **hdf5** format files, and utilize the **python** scripts provided in *Auxiliary Tool User Guide* for result analysis.

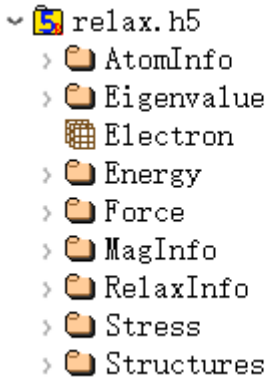
Except for the charge density file *rho.h5* and the solvent-bound charge density output file *rhoBound.h5*, the filenames of other output files depend on the **task** type. Currently, DS-PAW supports 14 task types, and the corresponding h5 filenames are: *relax.h5*, *scf.h5*, *band.h5*, *dos.h5*, *potential.h5*, *elf.h5*, *pcharge.h5*, *frequency.h5*, *elastic.h5*, *neb.h5*, *phonon.h5*, *aimd.h5*, *epsilon.h5*, and *wannier.h5*.

DS-PAW 2023A currently supports output files in **.json** format, but users are not advised to continue using this format for analysis. DS-PAW will completely remove the json format output during iterative versions, ceasing maintenance and updates for this format. Users can control whether to output json files via the `io.outJsonFile` parameter.

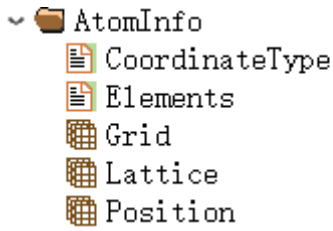
## 6.1 relax.h5

relax.h5 is the output file when task = relax; this file is not output when the task type is something else.

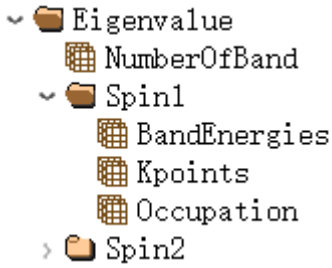
The *relax.h5* file contains at least 9 basic structures:



(1) *AtomInfo* saves the basic structural information of the system, such as cell size, atomic positions, etc.;

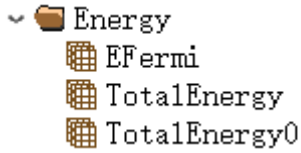



(2) *Eigenvalue* stores the number of bands calculated, spin information, the number of k-points and their coordinates, the orbital occupation numbers and energy eigenvalues of each band at each k-point;

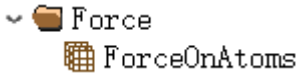



(3) *Electron* saves the total number of valence electrons in the system;

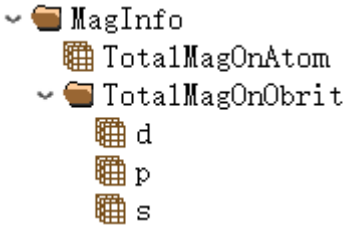
(4) *Energy* stores the total energy and Fermi energy;



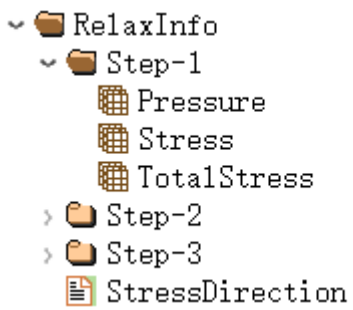

(5) The force on each atom during the relaxation process is saved in *Force*;



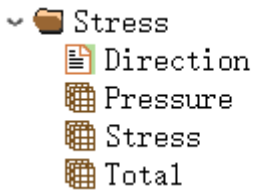

(6) *MagInfo* stores the total magnetic moment information of atoms; if projections are enabled, it stores the projected magnetic moment information.



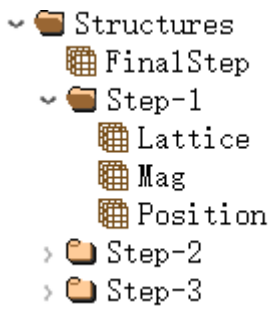

(7) The *RelaxInfo* saves the stress and pressure data of the system at each step during structural relaxation;




(8) *Stress* stores the stress magnitude in each direction of the unit cell, and the system pressure.



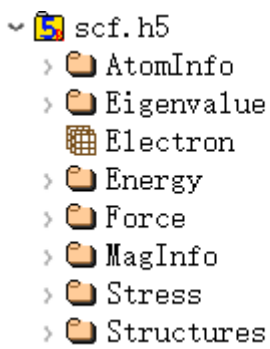

(9) Structures stores structure and magnetic moment data during relaxation;




## 6.2 scf.h5

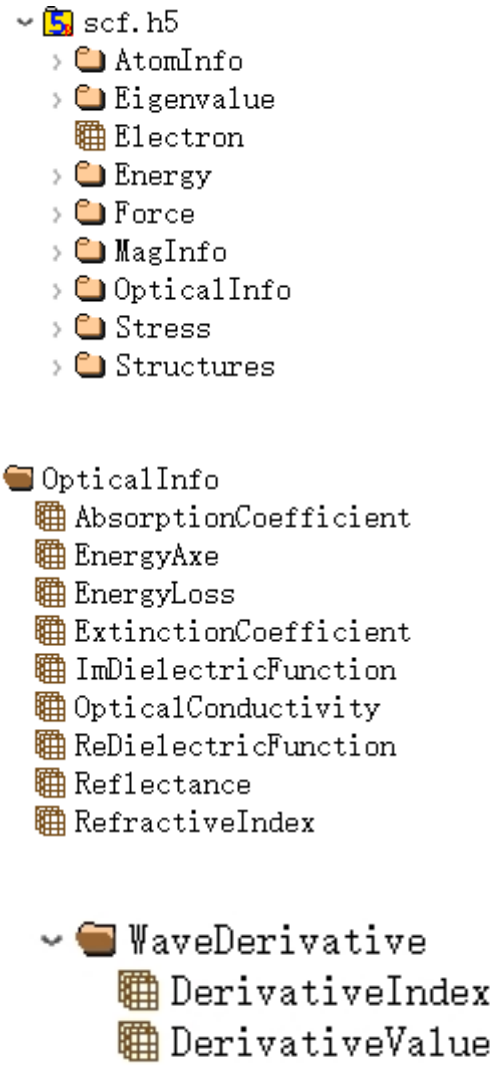
scf.h5 is the output file when task = scf; this file is not output for other task types.

The *scf.h5* file contains at least 8 fundamental structures, with basic information consistent with *relax.h5*:

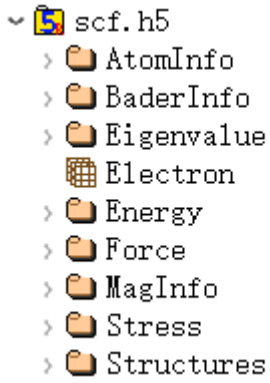


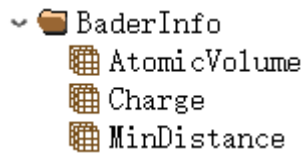

*Under *task = scf*, calculations for various functionalities can be controlled through parameters such as `sys` and `io`. The generated *scf.h5* file will store data corresponding to these functionalities. Specifically, the calculations can be categorized as follows:

(1) By setting `io.optical = true`, linear optical properties are calculated based on the self-consistent field (SCF) calculation:

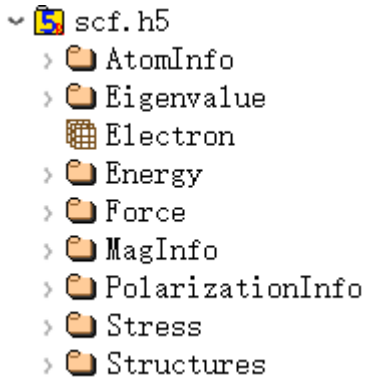


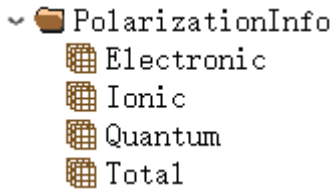

(2) Calculate Bader charge based on the self-consistent calculation by setting `io.bader = true`:



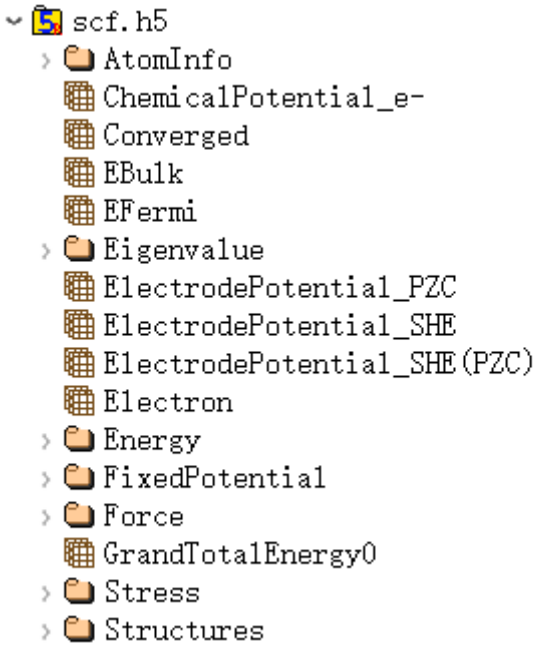



(3) Perform ferroelectric calculations based on the self-consistent calculation by setting `io.polarization = true`:



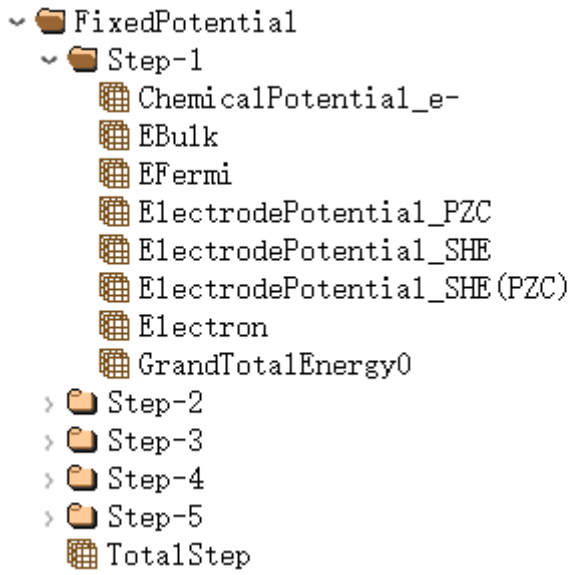




(4) Perform fixed potential calculations in the self-consistent calculation by setting `sys.fixedP = true`:



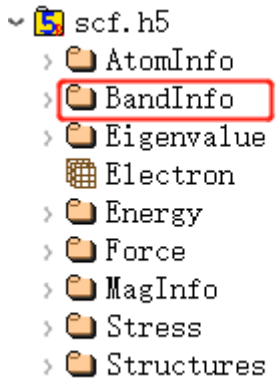

where *ChemicalPotential_e* is the electronic chemical potential of the system; *EBulk* is the negative value of

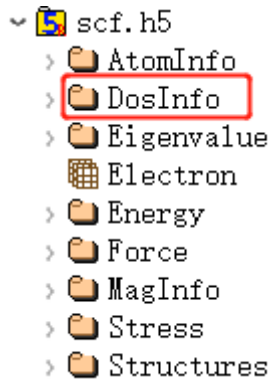
the Fermi level shift under the implicit solvent model; *ElectrodePotential* gives the potential value under different calibration standards; and *GrandTotalEnergy0* gives the total energy of the system under the grand canonical ensemble of electrons.

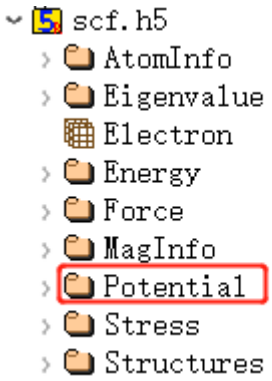


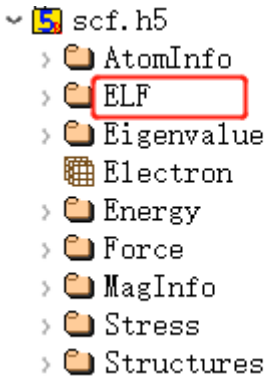

Expand under the *fixedPPotential* tag to summarize information on key parameters during the electronic iteration.

(5) **By setting `io.band = true`, `io.dos = true`, `io.potential = true`, and `io.elf = true`,**
Perform band structure calculation, density of states calculation, potential function calculation, and electron localization density calculation based on the self-consistent calculation:





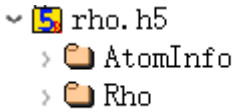




## 6.3 rho.h5

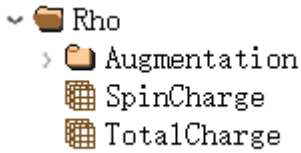
rho.h5 is the charge density output file for each task.

*rho.h5* contains two structures:




Where *AtomInfo* is consistent with the *AtomInfo* structure in the *relax.h5* file, and *Rho* stores the charge density

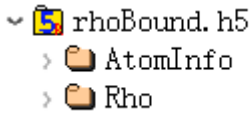
data:




## 6.4 rhoBound.h5

rhoBound.h5 is the output file for the solvent-bound charge density in the solvation model.

*rhoBound.h5* contains two structures:

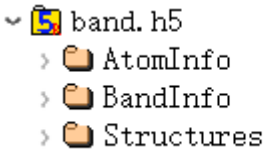



Where *AtomInfo* is largely consistent with the *AtomInfo* structure in the *relax.h5* file, and *Rho* stores the solvent-bound charge density data:

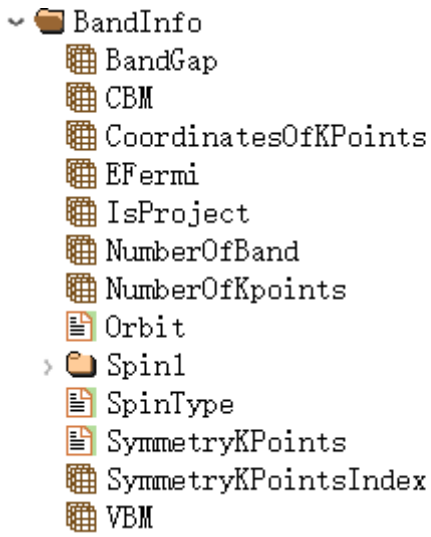
## 6.5 band.h5

band.h5 is the output file for each task = band. This file is not output when the task type is different.

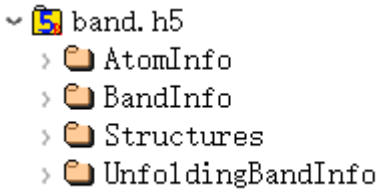
*band.h5* contains at least 3 structures:



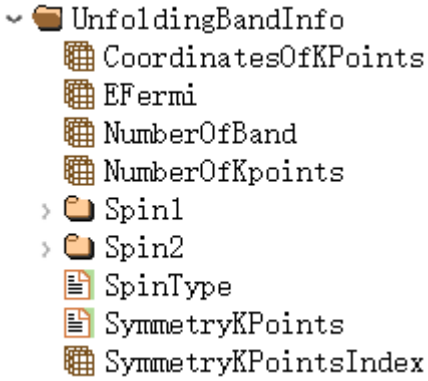

The structures of *AtomInfo*, *Structures*, and the *relax.h5* file are consistent. *BandInfo* stores the band structure data:




The band folding calculation corresponding to *band.h5* should contain at least 4 structures:



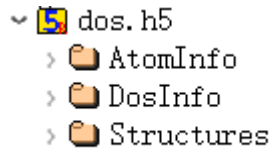

The structure of the structs corresponding to *AtomInfo*, *Structures*, and the *relax.h5* file are consistent. *BandInfo* stores band data, and *UnfoldingBandInfo* stores band unfolding data.



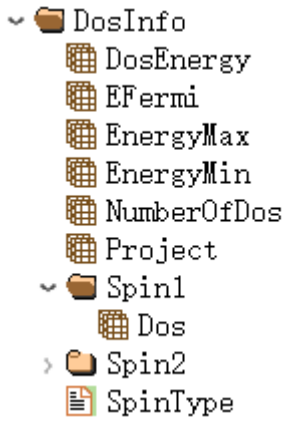
## 6.6 dos.h5

dos.h5 is the output file for task = dos; this file is not output when the task type is different.

*dos.h5* contains at least 3 structures:



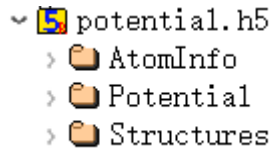

Among them, the structure of the structures corresponding to *AtomInfo*, *Structures*, and the file *relax.h5* are consistent, and *DosInfo* stores the density of states data:



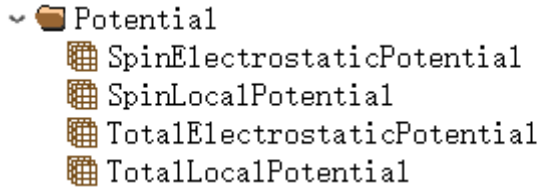

## 6.7 potential.h5

*potential.h5* is an output file under *task* = *potential*. This file is not generated for other task types.

The *potential.h5* file contains at least 3 structures:



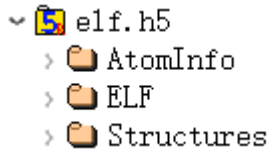

Where *Potential* stores the potential function data:



## 6.8 elf.h5

elf.h5 is the output file for task = elf. When the task type is different, this file is not output.

*elf.h5* contains at least 3 structures:

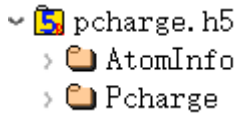



Where *elf* stores the local density data:

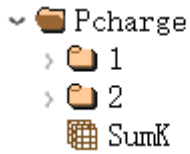

## 6.9 pcharge.h5

pcharge.h5 is the output file under task = pcharge; this file is not output when the task type is different.

The *pcharge.h5* file contains two structures:



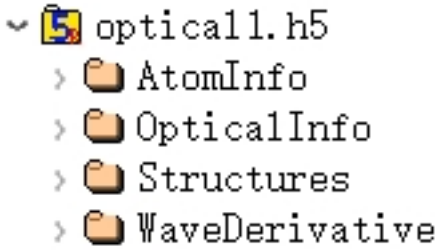

Here, *Pcharge* stores partial charge density data:



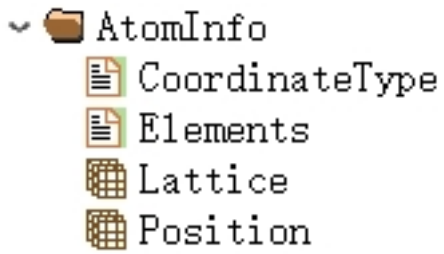

## 6.10 optical.h5

optical.h5 is the output file for task = optical; this file is not generated for other task types.

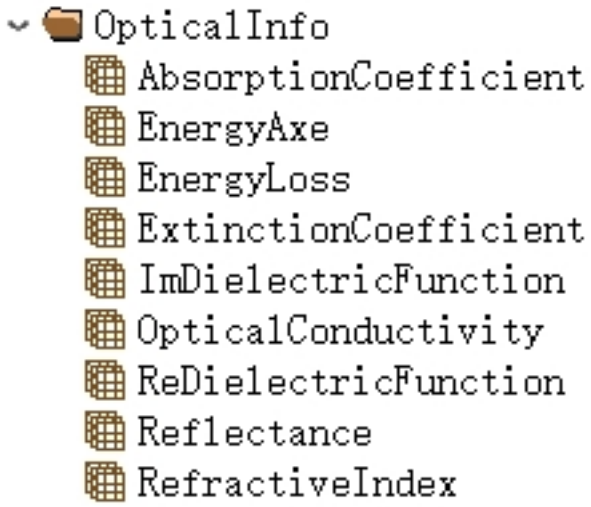
*optical.h5* contains four structures:



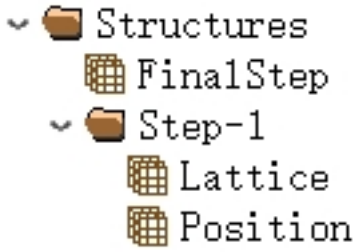
(1) Basic structural information of the system, such as unit cell size and atomic positions, is stored in *AtomInfo*:



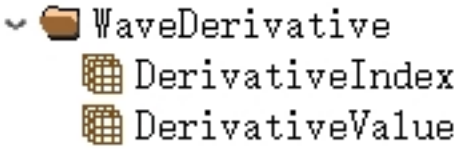

(2) The *opticalInfo* variable stores data on various properties from optical calculations:




(3) The optical calculation structure information is saved in *Structures*:



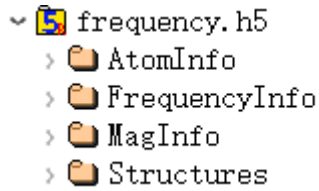

(4) *WaveDerivate* stores the derivative array of wave functions with respect to k-points, with a size of: (real part, imaginary part) * NumberOfBands (after selection) * NumberOfKPoints * NumberOfSpin * (x, y, z); *DerivativeIndex* gives the dimension of the derivative array; *DerivativeValue* gives the value of the derivative array.



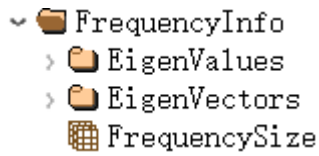
## 6.11 frequency.h5

*frequency.h5* is the output file for *task* = *frequency*; this file is not generated for other task types.

With spin considered, *frequency.h5* contains four structures:




Where frequency data is stored in *FrequencyInfo*:

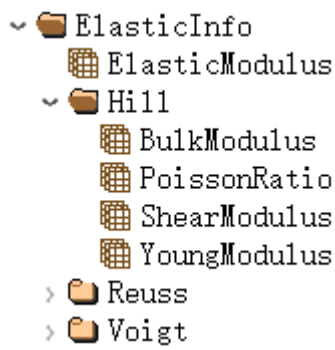



## 6.12 elastic.h5

*elastic.h5* is the output file for *task* = *elastic*. This file is not output when the *task* type is other than *elastic*.

With spin considered, *elastic.h5* contains four structures:

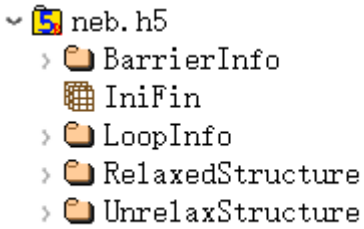

Elastic data is stored in *ElasticInfo*:



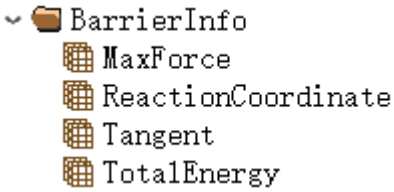

## 6.13 neb.h5

*neb.h5* is the output file in the top-level directory when *task* = *neb*.

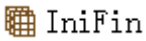
*neb.h5* contains five structures:



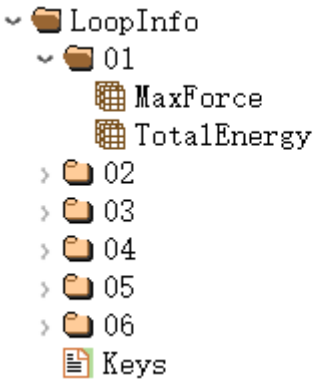

The *BarrierInfo* stores the maximum force, reaction coordinate (reaction distance between each image and the initial 00 structure), maximum shear force, and total energy data:



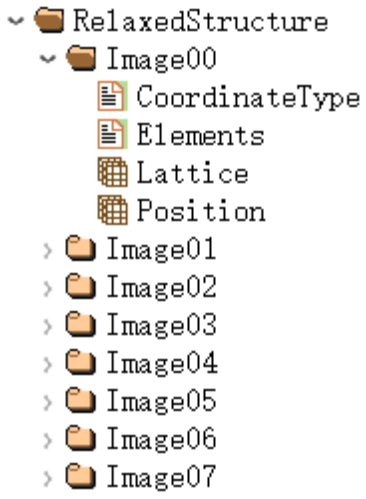

Where the switch for saving the initial and final states of the calculation is stored in *IniFin*:



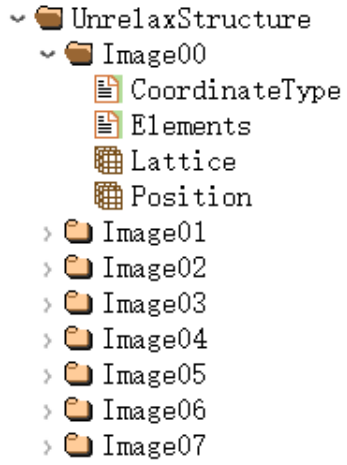

The *LoopInfo* object stores the energy and force changes for each image during the neb optimization process:




In which *RelaxedStructure* stores the structure data after optimization of each image:



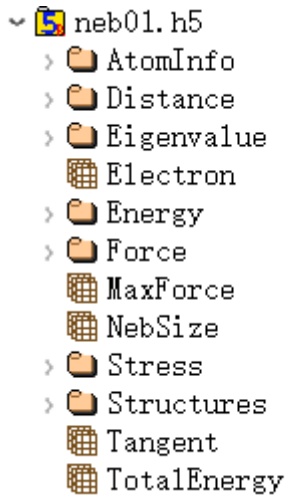

Where *UnrelaxStructure* stores the structural data before optimization of each image:




## 6.14 neb01.h5

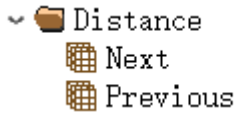
neb01.h5 is the output file in the 01 subdirectory when task = neb. Similarly, the neb02.h5 file will be generated in the 02 subdirectory.

In the spin-unpolarized case, *neb01.h5* contains 12 structures:




Among them, the structures of *AtomInfo*, *Eigenvalue*, *Electron*, *Energy*, *Force*, *Stress*, and *Structures* are consistent with the corresponding structure in the *relax.h5* file;

In the *Distance* data, the distance change of the reaction atom between the initial and final images during the optimization process is stored.




where *MaxForce* stores the maximum force data for image 1 during optimization;

Where *NebSize* stores the maximum number of steps in the transition state calculation.

where *Tangent* stores the data of the change in the tangent force of image 1 during the optimization process;

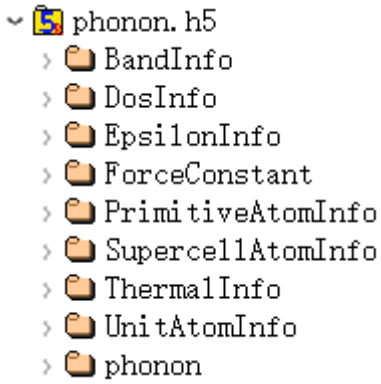
where *TotalEnergy* stores the total energy change data of image 1 during the optimization process;

## 6.15 phonon.h5

phonon.h5 is the output file under task = phonon. This file is not output when the task type is different.

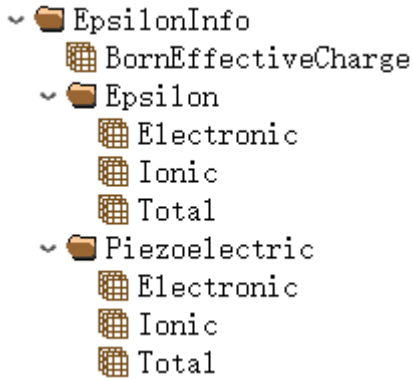
(1) When `phonon.method = dfpt`, the phonon.f5 file is as follows:

When the dfpt method is used to calculate phonon band structure and density of states, it enables the calculation of dielectric constants and phonon thermodynamics. The *phonon.h5* file contains 9 structures:

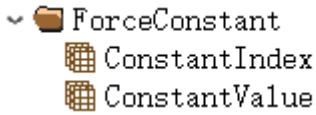



Where *BandInfo* and *DosInfo* respectively store the band structure and density of states data, their structures are consistent with the corresponding structures in *band.h5* and *dos.h5* files;

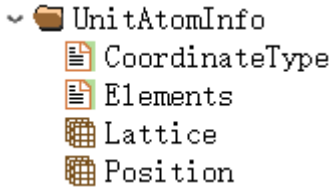
Where *EpsilonInfo* stores the dielectric function data:



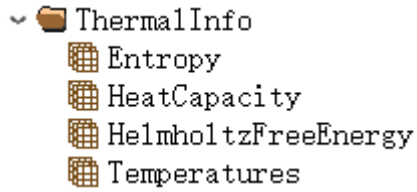

where the mechanical constant data is stored in *ForceConstant*:



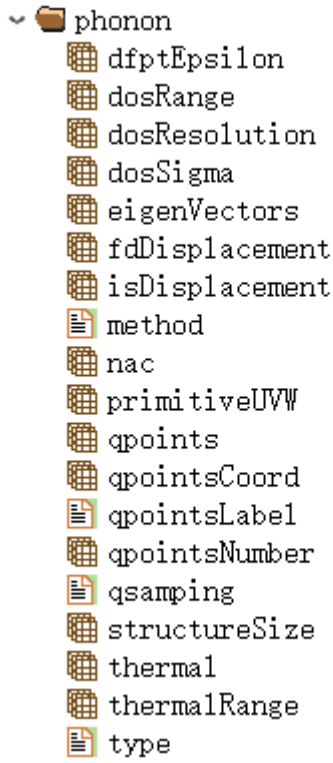

Where *PrimitiveAtomInfo*, *SupercellAtomInfo*, and *unitAtomInfo* store the structural information for the primitive cell, supercell, and unit cell, respectively. Taking the unit cell as an example, the structure is as follows:




In which *ThermalInfo* stores phonon thermodynamic data:

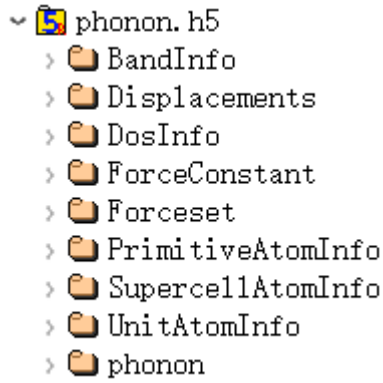



Here, *Phonon* stores the input parameters data for phonon calculations:




(2) When `phonon.method = fd`, the phonon.f5 file is shown as follows:

For phonon band structure and density of states calculations using the finite displacement method, *phonon.h5* contains 9 structures:

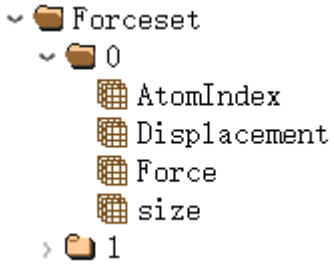



Where *BandInfo* and *DosInfo* store band structure and density of states data, respectively, whose structures corre-

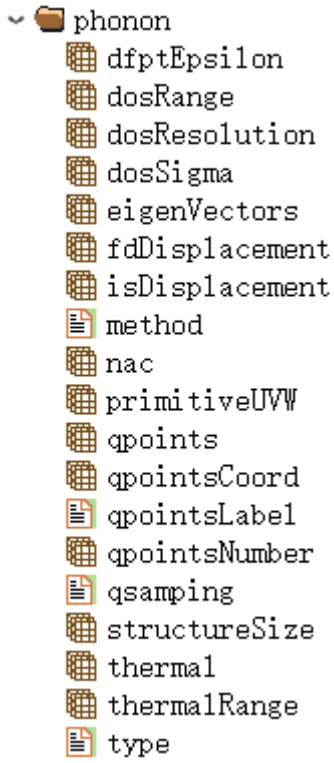
spond to the structures in the files *band.h5* and *dos.h5*, respectively.

where *ForceConstant* stores the force constant data, and *PrimitiveAtomInfo*, *SupercellAtomInfo*, and *unitAtomInfo* respectively saving the structural information of the primitive cell, supercell, and unit cell, whose structures are consistent with the structures in the `phonon.f5` file generated when `phonon.method = dfpt`;

Where *ForceSet* stores the mechanical matrix data calculated for each structure:



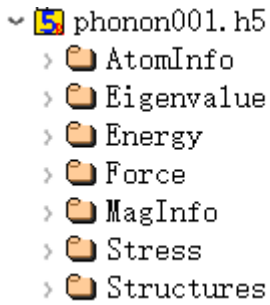

Here, the phonon calculation input parameter data is stored in *Phonon*:




## 6.16 phonon001.h5

task = phonon, when phonon.method = fd, the file phonon.h5 will be output under the 001 subfolder. This type of h5 file can be renamed to phonon001.h5. Similarly, the phonon.h5 file will also be generated under the 002 folder.

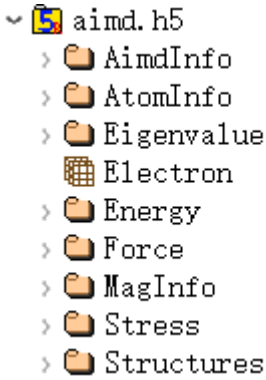
In the case of spin consideration, *phonon001.h5* contains 7 structures, as follows:




## 6.17 aimd.h5

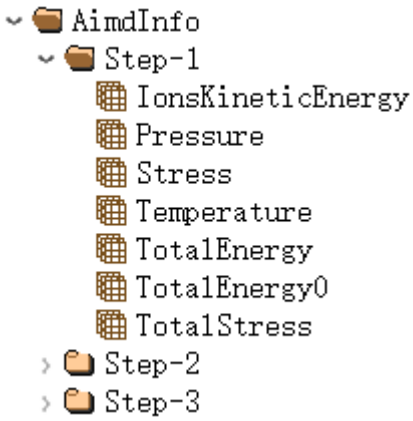
aimd.h5 is the output file for task = aimd; it is not output when the task type is different.

When considering spin, *aimd.h5* contains 9 structures:




The underlying structure information is consistent with *relax.h5*;

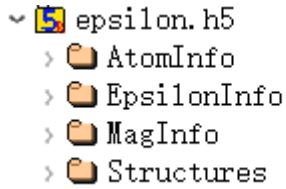
The new structure *AimdInfo* contains n structures, each storing the state information of the system under a certain ion step, such as temperature, pressure, energy, and kinetic energy:



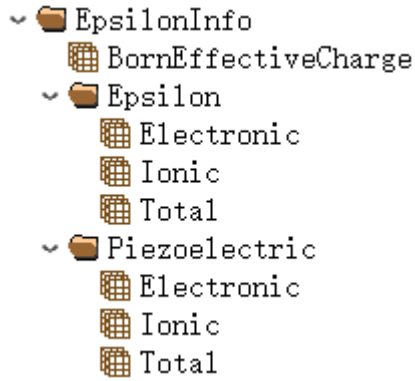
## 6.18 epsilon.h5

*epsilon.h5* is the output file for *task* = *epsilon*. This file is not produced for other task types

Considering spin, *epsilon.h5* contains four structures:



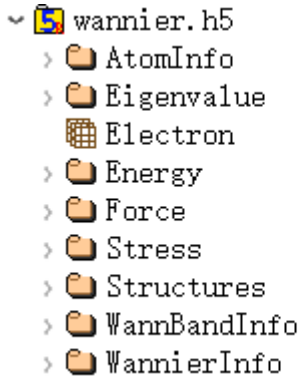

In which the dielectric constant data is stored in *EpsilonInfo*:




## 6.19 wannier.h5

wannier.h5 is the output file under task = wannier; this file is not output when the task type is other than wannier.

Without considering spin, the interpolated bands calculated by *wannier.h5* contain 9 structures:




In this, *WannBandInfo* stores the interpolated band data, which can be used to plot band diagrams:

- WannBandInfo
  - CoordinatesOfKPoints
  - EFermi
  - NumberOfBand
  - NumberOfKpoints
  - Spin1
    - BandEnergies
    - BandProject
  - SpinType
  - SymmetryKPoints
  - SymmetryKPointsIndex

Within this, *WannInfo* stores the Wannier function fitting data, including k-grid points and initial projection information, etc.:

- WannierInfo
  - BlochBandsKpoints
  - Projections
  - Spin1

In which, the *spin1* under *WannInfo* stores the fitting data of Wannier functions, including the calculated Hamiltonian and other data:

- Spin1
  - HaveDisentangled
  - Lattice
  - NumberOfBlochBands
  - NumberOfDFTBands
  - NumberOfWannierFunctions
  - OverlapMatrix
  - RecipLattice
  - RotationMatrix
  - Spreads
    - Invariants
    - Total
  - WannierFunctionCenters
  - WannierSpaceHamiltonian
    - ImaginaryHamiltonian
    - NumberOfRpoints
    - RealHamiltonian
    - RpointsDegeneracy
    - RpointsVector

# 7

# Restart calculation

DS-PAW currently supports resuming calculations for **structure relaxation**, **transition state calculations**, **molecular dynamics simulations**, **constant-potential calculations**, and **reading rho and wave** functions. Users can specify file paths to read the final structure, magnetic moments, potential, and other relevant information from the previous calculation.

## 7.1 Continuation of Relax Calculation Instructions

In the event that the relaxation calculation is unexpectedly terminated, fails to converge within the maximum number of steps, or if a higher-precision relaxation calculation is desired, it is necessary to obtain the final structure from the previous calculation (including the magnetic moment information of the final configuration if spin is considered in the system) to perform the next relaxation calculation. In this case, the program will output *latestStructure.as* and *relax.h5* files, both *latestStructure.as* and *relax.h5* can be used as input files for a subsequent calculation. If you need to continue the calculation based on this structure, it is recommended to follow these steps:

1. Create a clean directory and prepare two input files: *relax.in* and *latestStructure.as* (or *relax.h5*);
2. Set the parameter **sys.structure = latestStructure.as** (or **sys.structure = relax.h5**) in the *relax.in* file. The name of the structure file can be modified, and it is recommended to provide a clear indication for the continuation calculation.
3. Submit the job for calculation.

*latestStructure.as* is one of the readable files for structure relaxation continuation calculations. In addition to this, the *relax.h5* file can also be read as the final state structure.

## 7.2 NEB Transition State Calculation Continuation Instructions

If the transition state calculation is unexpectedly terminated, fails to converge within the maximum number of steps, or requires a higher accuracy calculation, you need to obtain the final structure from the previous calculation (including the magnetic moment information if the system considers spin) to perform the next transition state calculation. The transition state calculation involves multiple subfolders; in this case, In each subfolder `No`, the files *latestStructureNo.as* and *nebNo.h5* are output by default. The *.as* file can be used as the input file for a subsequent calculation.

Taking an insertion point number of **3** as an example, if you need to continue the calculation based on this structure, you can directly call the NEB continuation script described in *Auxiliary Tool User Guide*:

Demonstration of the data processing procedure using a Python script:

1. Enter the directory of the initial NEB calculation and view the files in that directory:

```
(base) [hzw1002@mgt2 neb]$ ls
00  01  02  03  04  dev.slurm  DS-PAW.log  _err.dat  input.in  neb-restart.py  _out.dat
```

2. Call the *:guilabel: `neb_restart.py* script in this directory and execute the following command:

```
python neb_restart.py
```

Following the prompts in the interactive interface, specify the path to the original NEB file, the parameter file name, and the backup folder name. In this example, the backup folder is specified as *bakfile*.

3. Check the neb directory again:

```
(base) [hzw1002@mgt2 neb]$ ls
00  01  02  03  04  bakfile  dev.slurm  _err.dat  input.in  neb-restart.py  _out.dat
```

Where *bakfile* is the backup file, and the *00-04* folders store the structure files required for resuming the calculation. You can submit directly within this directory to resume the calculation.

4. Backup folder *bakfile* structure analysis.

```
.
├── 00
│   ├── 00.tar.xz
│   ├── latestStructure00.as
│   └── structure00.as
├── 01
│   ├── 01.tar.xz
│   ├── latestStructure01.as
│   └── structure01.as
├── 02
│   ├── 02.tar.xz
│   ├── latestStructure02.as
│   └── structure02.as
├── 03
│   ├── 03.tar.xz
│   ├── latestStructure03.as
│   └── structure03.as
├── 04
│   ├── 04.tar.xz
│   ├── latestStructure04.as
│   └── structure04.as
├── DS-PAW.log
└── neb.tar.xz
```

The outermost compressed archive, *neb.tar.xz*, in the backup folder contains the initial NEB calculation *h5* files. The compressed archives in each subfolder contain backups of all files from the corresponding subfolder of the initial NEB calculation. The outermost layer outside the subfolders contains the initial and final state structure files from the initial calculation.

If users prepare input files themselves, it is recommended to follow the steps below:

1. Create a clean directory and place the *neb.in* file, initial and final structure files *structure00.as* and *structure04.as*, and the final structure files for intermediate configurations *latestStructure01.as*, *latestStructure02.as*, and *latestStructure03.as* into it;
2. Rename the intermediate structure files, *:guilabel: `latestStructureNo.as*, to *:guilabel: `structureNo.as*;
3. Create folders **00**, **01**, **02**, **03**, and **04**, and place the corresponding structure files in each folder;
4. Submit the job for calculation.

The *.as* file is a readable file for continuing transition state calculations; it is not recommended to use the *nebNo.h5* file as input for continuing calculations.

## 7.3 Instructions for Continuing AIMD Molecular Dynamics Simulations

If the molecular dynamics simulation was unexpectedly terminated, or if you wish to extend the simulation time, you need to obtain the final structure and velocities (and magnetization information for spin-polarized systems) from the previous calculation to perform a longer simulation. The molecular dynamics simulation by default outputs *latestStructure.as* and *aimd.h5* file. Both *latestStructure.as* and *aimd.h5* can be used as input files for continuation. If you need to continue the calculation from this structure, it is recommended to follow the steps below:

1. Create a clean directory and prepare two input files: *aimd.in* and *latestStructure.as* (or *aimd.h5*);
2. In the *aimd.in* file, set the parameter **sys.structure = latestStructure.as** (or **sys.structure = aimd.h5**). The name of the structure file can be modified; it is recommended to include a clear continuation run indication in the filename.
3. Submit the job for calculation.

*latestStructure.as* is one of the readable files for continuing molecular dynamics calculations, and besides that, the *aimd.h5* file can also be read as the final structure.

**Note**

1. To modify the ensemble for a continuation run, delete the information in the **Next positions** section of the *latestStructure.as* file; otherwise, the continuation run may result in errors.

## 7.4 Continuation Instructions for fixedPotential Calculations

The fixedPotential calculation uses the steepest descent method, which solves the target charge and potential values through multiple self-consistent iterations. This process can be viewed as n self-consistent calculations that depend on each other. If the calculation is unexpectedly interrupted before the charge converges, the continuation function can be used. This function uses the charge and potential values obtained before the interruption as the starting point to approach the target potential. The following steps are recommended to resume a constant potential calculation:

1. In the original calculation directory, modify the *fixedPotential.in* file to specify the directory containing the h5 file obtained from the initial calculation to resume the calculation. The corresponding parameter is `cal.iniFixedP = ./scf.h5`.

**Note**

1. If you want to keep the scf.h5 file from the initial calculation, you can rename the original file, such as renaming it to readscf.h5, and set *cal.iniFixedP = ./readscf.h5*.
2. When continuing a calculation, the number of electrons and the target electrode potential are obtained from the specified file; modifying these parameters in the *in* file will have no effect.

## 7.5 Read rho and wave restart instructions

Due to the computational expense of hybrid functional calculations, when a calculation fails to converge in one step or when higher convergence accuracy is desired, the already obtained charge density and wave function files can be read. This is achieved by specifying the file paths using the `cal.iniCharge` and `cal.iniWave` parameters. The following `Resatrt-HSE.in` file lists the key parameters for restarting a hybrid functional calculation:

```
# task type
task = scf

#hybrid related
sys.hybrid=true
sys.hybridType=HSE06

#read related
cal.iniCharge = ../01/rho.bin
cal.iniWave = ../01/wave.bin

#outputs related
io.charge = true
io.wave = true
```

**Note**

1. For hybrid functional calculations, both the charge density and wave function files are required for continuation, and neither can be omitted.
2. For hybrid functional calculations, it is recommended to output the rho.bin and wave.bin files, which can be used as input for continuation calculations.

# 8

# Auxiliary Tool User Guide

**Note**

Want to quickly analyze results, plot data, or perform common data processing tasks after completing a DFT calculation with DSPAW?

`dspawpy` (Python >= 3.9) is such a tool. It can be called programmatically (see example scripts below) and also provides a command-line interactive program.

After following the tutorial *installation instructions*, you can use the interactive program by typing `dspawpy` and pressing Enter in the command line:

```
 ... loading dspawpy cli ...

 This is the dspawpy command-line interactive tool.  Enjoy!
    ( )
   _| |  ___  _ _      _ _  _   _   _  _ _    _   _
 /'_` |/',__)( '_`\  /'_` )( ) ( ) ( )( '_`\ ( ) ( )
( (_| |\__, \| (_) )( (_| || \_/ \_/ || (_) )| (_) |
 \__,_)(____/| ,__/'`\__,_) \___x___/ | ,__/  \__, |
             | |                      | |    ( )_| |
             (_)                      (_)     \___/

 Version: Installation Path

 ======================================
 |  1: Update
 |  2: structure conversion
 |  3: Volumetric data processing
 |  4: Band structure calculation
 |  5: Density of States (DOS) data processing
 |  6: Joint display of band structure and density of states (DOS)
 |  7: Optical properties data processing
 |  8: NEB (Nudged Elastic Band) transition state calculation data processing
```

```
|  9: phonon calculation data processing
|  10: aimd ab initio molecular dynamics data processing
|  11: Polarization data processing
|  12: ZPE zero-point energy data processing
|  13: Thermal correction energy of TS
|
|  q: Quit
=====================================
--> Enter a number and press Enter to select a function:
```

Highlights:

- Autocompletion: Works by pressing the Tab key, helping to quickly and correctly enter the required program arguments.
- Multithreaded lazy loading: Loads modules in the background while waiting for user input, significantly reducing waiting time; loads only necessary modules, minimizing memory usage.

Note:

- When using on a remote server, the startup time may be longer due to poor disk I/O performance, potentially taking up to half a minute in extreme cases (directly related to the servers current disk I/O performance). If this is unacceptable, please install and use dspawpy on your own computer.
- After typing *dspawpy* and pressing Enter, Python will first load built-in modules. Once this is complete, the prompt loading dspawpy cli will appear, indicating the second stage (loading third-party dependencies) has begun.
- After the second stage is completed, a welcome screen will be displayed, indicating that dspwapy has finished the initial loading and has entered the third stage. Subsequently, it will dynamically load the corresponding dependency libraries based on the selected functional modules, thereby minimizing waiting time.

## 8.1 Installation and Updates

1. On the HZW machine, dspawpy has been pre-installed. Activate the virtual environment using the following command to start using it:

```
source /data/hzwtech/profile/dspawpy.env
```

2. On other machines, please install dspawpy yourself (choose one of the following two methods):

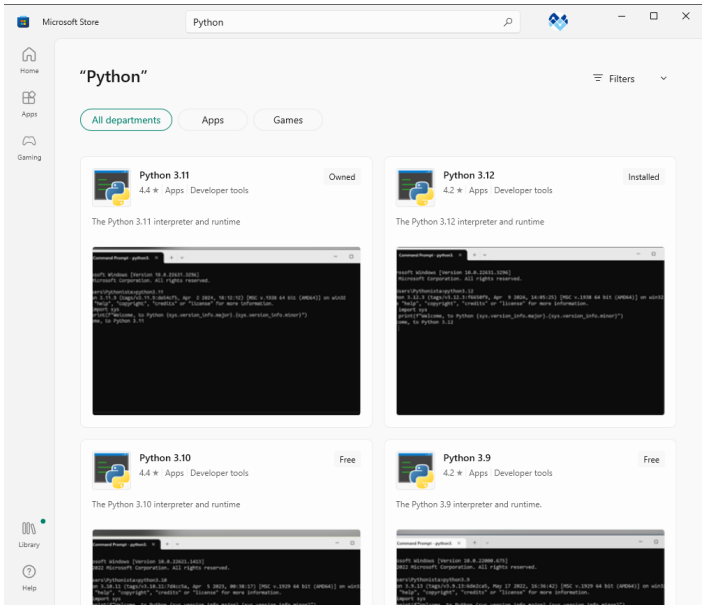
- Using mamba or conda, you can install the package from https://conda-forge.org/download/.

```
mamba install dspawpy -c conda-forge
#conda install dspawpy -c conda-forge
```

- Or, use pip3 (some operating systems may not have the executable *pip3*, in which case try *pip*)

### pip

- pip3 is the package manager that comes with python3.
- Linux and Mac usually come with Python 3 and pip3.
- On Windows, open the Microsoft Store, search for Python, and install it.




Then open cmd or powershell to use pip.

```
pip3 install dspawpy
```

For information on how to configure pip and conda mirror addresses to speed up the installation process, please refer to https://mirrors.tuna.tsinghua.edu.cn/help/pypi/ and https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/.

If the installation still fails, try the mamba/conda installation method above.

**Warning**

On clusters, due to permission issues, the *pip* in the public path may not support global installation of Python libraries. You must add the *–user* option after *pip install* to install them in your home directory under *~/.local/lib/python3.x/site-packages/*, where 3.x represents the Python interpreter version, and x can be any integer between 9 and 13.

Python will prioritize loading dspawpy from the home directory, even if the version in the public environment is newer! Therefore, if you have previously installed dspawpy with `--user` and have forgotten to manually update the old version in your home directory, even after sourcing the public environment, you will not be able to call the dspawpy in the public environment. Instead, the old version will still be used, leading to some bugs.

Therefore, **considering that the HZW cluster automatically updates dspawpy weekly, it is recommended not to install it redundantly in your home directory; delete any existing installations.** On other clusters, ensure that you manually update dspawpy in your home directory in a timely manner.

If you prefer not to delete and update the *dspawpy* in your home directory, you can use the *-s* option when running your Python scripts to prevent importing *dspawpy* from your home directory: `python -s your-script.py`.

### 8.1.1 Update dspawpy

To update dspawpy if it was installed with mamba/conda, use the following command:

```
mamba update dspawpy
#conda update dspawpy
```

If dspawpy was installed via pip:

```
pip install dspawpy -U # -U for upgrading to the latest version
```

**Note**

If pip uses a domestic mirror site, it may fail to upgrade smoothly because the mirror site has not yet synchronized the latest version of dspawpy dspawpy. Please use the following command to tell pip to download and install from the official PyPI site:

```
pip install dspawpy -i https://pypi.org/simple --user -U # -i specifies the download␣
↪address, --user installs for the current user only, and -U installs the latest␣
↪version
```

If you encounter errors related to **dspawpy** during runtime, first verify that you have correctly imported the latest version of dspawpy and check the installation path:

```
$ python3 # or python

>>> import dspawpy
>>> dspawpy.__version__ # will output the version number
>>> dspawpy.__file__ # will output the installation path
```

## 8.2 structure structure conversion

To read structure information, use the `read` function; to write structure information to a file, use the `write` function; for quick structure conversion, use the `convert` function:

### API: read(), write(), convert()

`dspawpy.io.structure.convert`(*infile*, *si=None*, *ele=None*, *ai=None*, *infmt: str | None = None*, *task: str = 'scf'*, *outfile: str = 'temp.xyz'*, *outfmt: str | None = None*, *coords_are_cartesian: bool = True*)

convert from infile to outfile.

- multi -> single, only keep last step
- crystal -> molecule, will lose lattice info
- molecule -> crystal, will add a box of twice the maximum xyz
- pdb, dump may suffer decimal precision loss

**Parameters**

- **infile** –
  - h5/json/as/hzw/cif/poscar/cssr/xsf/mcsqs/prismatic/yaml/fleur-inpgen file path
  - If a folder is given, will read {task}.h5/json files
  - If structures are given, will read multiple structures.
- **si** (*int, list, or str*) –
  - Structure index, starting from 1
    - si=1, read the 1st

    * si=[1,2], read the 1st and 2nd
    * si=:, read all
    * si=-3:, read the last 3
  - If empty, for multi-configuration files, all configurations will be read; for single-configuration files, the latest configuration will be read.
  - This parameter is only valid for h5/json files.
- **ele** –
  - Element symbol, written as H or [H,O]
  - If empty, atomic information for all elements will be read.
  - This parameter is only valid for h5/json files.
- **ai** –
  - Atom index, starting from 1
  - Usage is the same as si
  - If empty, atomic information for all atoms will be read.
  - This parameter is only valid for h5/json files.
- **infmt** –
  - Input structure file type, e.g., h5. If None, the file extension will determine the format.
- **task** –
  - Used when datafile is a folder path to locate the internal {task}.h5/json file.
  - Calculation task type, including scf, relax, neb, aimd. Other values will be ignored.
- **outfile** –
  - Output filename
- **outfmt** –
  - Output structure file type, e.g., xyz. If None, the file extension will determine the format.
- **coords_are_cartesian** –
  - Whether to write coordinates in Cartesian form (default: True); otherwise, fractional coordinates will be used.
  - This option is currently only valid for as and json formats.

### Examples

```
>>> from dspawpy.io.structure import convert
>>> convert('dspawpy_proj/dspawpy_tests/inputs/supplement/PtH.as', outfile='dspawpy_
↪proj/dspawpy_tests/outputs/doctest/PtH.hzw')
==> ...PtH.hzw...
```

batch test

```
>>> for readable in ['relax.h5', 'system.json', 'aimd.pdb', 'latestStructure.as',
↪'CuO.hzw', 'POSCAR']:
...     for writable in ['pdb', 'xyz', 'dump', 'as', 'hzw', 'POSCAR']:
...         convert('dspawpy_proj/dspawpy_tests/inputs/supplement/stru/'+readable,␣
↪outfile=f"dspawpy_proj/dspawpy_tests/outputs/doctest/{readable.split('.')[0]}.
↪{writable}")
==> ...relax.pdb...
==> ...relax.xyz...
==> ...relax.dump...
==> ...relax.as...
==> ...relax.hzw...
==> ...system.pdb...
==> ...system.xyz...
==> ...system.dump...
==> ...system.as...
==> ...system.hzw...
==> ...aimd.pdb...
==> ...aimd.xyz...
==> ...aimd.dump...
==> ...aimd.as...
==> ...aimd.hzw...
==> ...latestStructure.pdb...
==> ...latestStructure.xyz...
==> ...latestStructure.dump...
==> ...latestStructure.as...
==> ...latestStructure.hzw...
==> ...CuO.pdb...
==> ...CuO.xyz...
==> ...CuO.dump...
==> ...CuO.as...
==> ...CuO.hzw...
==> ...POSCAR.pdb...
==> ...POSCAR.xyz...
==> ...POSCAR.dump...
==> ...POSCAR.as...
==> ...POSCAR.hzw...
```

`dspawpy.io.structure.read`(*datafile: str | list*, *si=None*, *ele=None*, *ai=None*, *fmt: str | None = None*, *task: str | None = 'scf'*)

Read one or more h5/json files and return a list of pymatgen Structures.

**Parameters**

- **datafile** –
  - file paths for h5/json/as/hzw/cif/poscar/cssr/xsf/mcsqs/prismatic/yaml/fleur-inpgen files;
  - If a directory path is given, it can be combined with the task parameter to read the {task}.h5/json files inside
  - If a list of strings is given, it will sequentially read the data and merge them into a list of Structures
- **si** (*int, list or str*) –
  - Configuration number, starting from 1

    * si=1, reads the first configuration
    * si=[1,2], reads the first and second configurations
    * si=:, reads all configurations
    * si=-3:, reads the last three configurations
  - If empty, it reads all configurations for multi-configuration files and the latest configuration for single-configuration files
  - This parameter is only valid for h5/json files
- **ele** –
  - Element symbol, format reference: H or [H,O]
  - If empty, it will read atomic information for all elements
  - This parameter is only valid for h5/json files
- **ai** –
  - Atom index, starting from 1
  - Same as si
  - If empty, it will read all atom information
  - This parameter is only valid for h5/json files
- **fmt** –
  - File format, including as, hzw, xyz, pdb, h5, json 6 types, other values will be ignored.
  - If empty, the file type will be determined based on file name conventions.
- **task** –
  - Used when datafile is a directory path to find the internal {task}.h5/json file.
  - Determine the task type, including scf, relax, neb, aimd four types, other values will be ignored.

**Returns**
Structure list

**Return type**
pymatgen_Structures

### Examples

```
>>> from dspawpy.io.structure import read
```

Reads a single file to generate a list of Structures

```
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪supplement/PtH.as')
>>> len(pymatgen_Structures)
1
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪supplement/PtH.hzw')
>>> len(pymatgen_Structures)
1
```

(continues on next page)

(continued from previous page)

```
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪supplement/Si2.xyz')
>>> len(pymatgen_Structures)
1
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪supplement/aimd.pdb')
>>> len(pymatgen_Structures)
1000
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/2.1/
↪relax.h5')
>>> len(pymatgen_Structures)
3
>>> pymatgen_Structures = read(datafile='dspawpy_proj/dspawpy_tests/inputs/2.1/
↪relax.json')
>>> len(pymatgen_Structures)
3
```

Note that pymatgen_Structures is a list composed of multiple Structure objects, each corresponding to a structure. If there is only one structure, it will also return a list. Please use pymatgen_Structures[0] to obtain the Structure object.

When datafile is a list, it reads multiple files sequentially and merges them into a Structures list

```
>>> pymatgen_Structures = read(datafile=['dspawpy_proj/dspawpy_tests/inputs/
↪supplement/aimd1.h5','dspawpy_proj/dspawpy_tests/inputs/supplement/aimd2.h5'])
```

`dspawpy.io.structure.write`(*structure*, *filename: str*, *fmt: str | None = None*, *coords_are_cartesian: bool = True*)

Write information to the structure file

**Parameters**

- **structure** – *A pymatgen Structure object*
- **filename** – Structure filename
- **fmt** –
  - Structure file type, natively supports json, as, hzw, pdb, xyz, dump six types
- **coords_are_cartesian** –
  - Whether to write in Cartesian coordinates, default is True; otherwise write in fractional coordinate format
  - This option is currently only effective for as and json formats

### Examples

First, read the structure information:

```
>>> from dspawpy.io.structure import read
>>> s = read('dspawpy_proj/dspawpy_tests/inputs/2.15/01/neb01.h5')
>>> len(s)
17
```

Writing structure information to a file:

```
>>> from dspawpy.io.structure import write
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.json', coords_
↪are_cartesian=True)
==> ...PtH.json...
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.as', coords_
↪are_cartesian=True)
==> ...PtH.as...
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.hzw', coords_
↪are_cartesian=True)
==> ...PtH.hzw...
```

PDB, XYZ, and DUMP file types can write multiple conformations to form a trajectory. The generated XYZ trajectory files can be opened and visualized using visualization software like OVITO.

```
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.pdb', coords_
↪are_cartesian=True)
==> ...PtH.pdb...
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.xyz', coords_
↪are_cartesian=True)
==> ...PtH.xyz...
>>> write(s, filename='dspawpy_proj/dspawpy_tests/outputs/doctest/PtH.dump', coords_
↪are_cartesian=True)
==> ...PtH.dump...
```

The recommended format for storing single structure information is as format. If the Structure contains magnetic moment or degree of freedom information, it will be written in the most complete format, such as Fix_x, Fix_y, Fix_z, Mag_x, Mag_y, Mag_z. The default value for degree of freedom information is F, and the default value for magnetic moment is 0.0. You can manually delete this default information from the generated as file as needed. Writing to other types of structure files will ignore magnetic moment and degree of freedom information.

See the `2conversion.py` script for conversion:

```
# coding:utf-8
from dspawpy.io.structure import convert

convert(
    infile="dspawpy_proj/dspawpy_tests/inputs/2.1/relax.h5",  # Structure to be␣
↪converted, if in the current path, you can just write the filename
    si=None,  # Select configuration number, if not specified, read all by default
    ele=None,  # Filter element symbol, default reads atomic information for all elements
    ai=None,  # Filter atomic indices, starting from 1, default to read all atomic␣
↪information
    infmt=None,  # Input structure file type, e.g., 'h5'. If None, it will be matched␣
↪ambiguously based on the filename rule.
    task="relax",  # Task type, this parameter is only valid when infile is a folder␣
↪rather than a filename
    outfile="dspawpy_proj/dspawpy_tests/outputs/us/relaxed.xyz",  # Structure file name
    outfmt=None,  # Output structure file type, e.g., 'xyz'. If None, it will be fuzzy␣
↪matched according to filename rules.
    coords_are_cartesian=True,  # Written in Cartesian coordinates by default
)
```

The rules for setting several key parameters of the convert function are shown in the table below:

Table 1: **dspawpy Supported IO format**

| infmt (input file format) | infile (Input file name fuzzy match) | outfmt (output file format) | outfile (output filename fuzzy match) | Description |
|---|---|---|---|---|
| h5 | **.h5* | X | X | HDF5 files saved after DS-PAW calculations are completed |
| json | **.json* | json | **.json* | json files saved after DS-PAW calculations are completed |
| pdb | **.pdb* | pdb | **.pdb* | Protein Data Bank |
| as | **.as* | as | **.as* | DS-PAW structure file containing atomic coordinates and other information |
| hzw | **.hzw* | hzw | **.hzw* | DeviceStudios default structure file |
| xyz | **.xyz* | xyz | **.xyz* | Supports only single conformation of molecular structure when reading, and extended-xyz type trajectory files including unit cell when writing |
| X | X | dump | **.dump* | LAMMPS dump-type trajectory files |
| X | **.cif*/*.mcif** | cif/mcif | **.cif*/*.mcif** | Crystallographic Information File |
| X | **POSCAR*/*CONTCAR* | poscar | **POSCAR** | VASP files |
| X | **.cssr** | cssr | **.cssr** | Crystal Structure Standard Representation |
| X | **.yaml/*.yml* | yaml/yml | **.yaml/*.yml* | YAML Aint Markup Language |
| X | **.xsf** | xsf | **.xsf** | eXtended Structural Format |
| X | **rnd-str.in*/*lat.in*/*bestsqs** | mcsqs | **rnd-str.in*/*lat.in** | Monte Carlo Special Quasirandom Structure |
| X | *inp*.xml/*.in*/inp_** | fleur-inpgen | **.in** | FLEUR structure file, requires the additional installation of the pymatgen-io-fleur library |
| X | **.res* | res | **.res* | ShelX res structure file |
| X | **.config*/*.pwmat** | pwmat | **.config/*.pwmat* | PWmat files |
| X | X | prismatic | **prismatic** | A file format used for STEM simulations |
| X | *CTRL** | X | X | Stuttgart LMTO-ASA files |

**Note**

- In the table above, * represents any character, and *X* indicates unsupported formats.
- h5, json, pdb, xyz, dump, and CONTCAR formats support trajectory information consisting of multiple structures (common in structure optimization, NEB, or AIMD tasks)
- The *in(out)fmt* parameter has higher priority than filename wildcard matching; for example, specifying *in(out)fmt=h5* allows any filename, even *a.json*.
- When writing structural information in *json* format, only visualization of NEB chain tasks is supported. See *Observing the NEB Chain* for details.
- Structure information from DS-PAW output files such as neb.h5, phonon.h5, phonon.json, neb.json, and wannier.json is currently not readable.

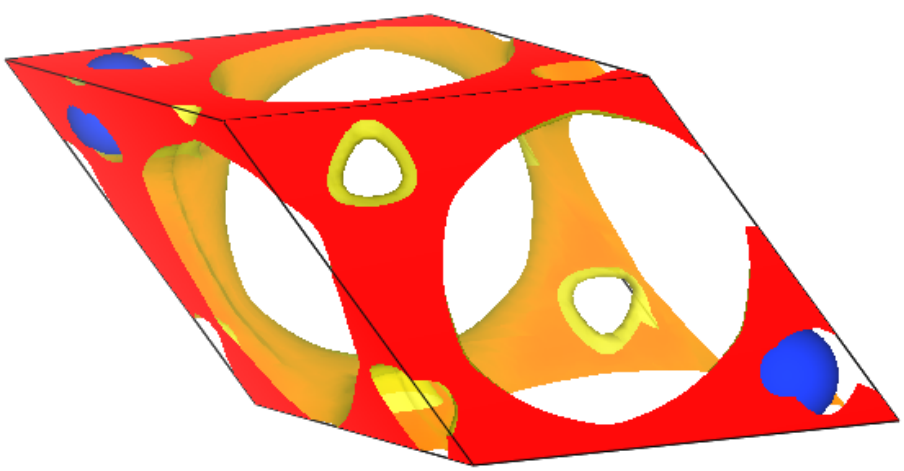
## 8.3 Volumetric Data Processing

### 8.3.1 volumetricData Visualization

- See also `3vis_vol.py`:

```
# coding:utf-8
from dspawpy.io.write import write_VESTA

# Read data file (in h5 or json format), process it, and output to a cube file
write_VESTA(
    in_filename="dspawpy_proj/dspawpy_tests/inputs/2.2/rho.h5",  # Path to the json␣
↪or h5 file containing electronic system information
    data_type="rho",  # Data type, supported values are "rho", "potential", "elf",
↪"pcharge", "rhoBound"
    out_filename="dspawpy_proj/dspawpy_tests/outputs/us/DS-PAW_rho.cube",  # Output␣
↪file path
    gridsize=(10, 10, 10),  # Specifies the interpolation grid size
    format="cube",  # Supported formats: cube, vesta, and txt (xyz grid coordinates␣
↪+ values)
)
```

Drag the converted file *DS-PAW_rho.cube* into the **VESTA** software to visualize it:



### 8.3.2 Differential volumetric data visualization

- See `3dvol.py`:

```
# coding:utf-8
from dspawpy.io.write import write_delta_rho_vesta

# Read the data file (h5 or json format), process it, and output it to a cube file,␣
↪which can be directly opened with Vesta and has a small volume
write_delta_rho_vesta(
```

(continues on next page)

(continued from previous page)

```
    total="dspawpy_proj/dspawpy_tests/inputs/supplement/AB.h5",  # Data file for the system containing all components
    individuals=[
        "dspawpy_proj/dspawpy_tests/inputs/supplement/A.h5",
        "dspawpy_proj/dspawpy_tests/inputs/supplement/B.h5",
    ],  # Data files for the system containing each component
    output="dspawpy_proj/dspawpy_tests/outputs/us/3delta_rho.cube",  # Output file path
)
```

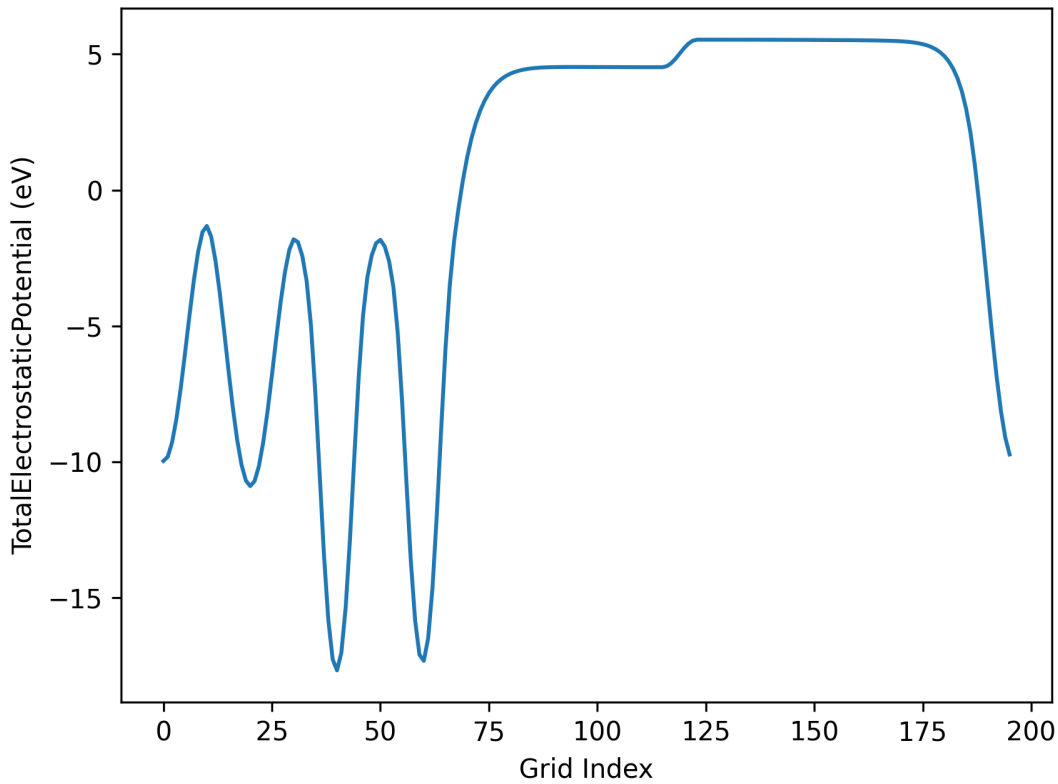
The above script supports the processing of charge density differences in multi-component systems. As an example using a binary system, it generates the charge density difference file *delta_rho.cube* from AB.h5, A.h5, and B.h5. This file can be directly opened using **VESTA**.

### 8.3.3 Volumetric data plane average

- See also `3planar_ave.py`:

```
# coding:utf-8
from dspawpy.plot import average_along_axis

axes = [
    "2"
]  # "0", "1", "2" correspond to the x, y, z axes respectively; select which axes to average along
axes_indices = [int(i) for i in axes]
for ai in axes_indices:
    plt = average_along_axis(
        datafile="dspawpy_proj/dspawpy_tests/inputs/3.3/scf.h5",  # Data file path
        task="potential",  # Task name, can be 'rho', 'potential', 'elf', 'pcharge', 'rhoBound'
        axis=ai,  # Axis along which to plot the potential curve
        smooth=False,  # Whether to smooth
        smooth_frac=0.8,  # Smoothing coefficient
        subtype=None,  # Used to specify the subclass of task data, currently only used for Potential
        label=f"axis{ai}",  # Legend label
    )
if len(axes_indices) > 1:
    plt.legend()

plt.xlabel("Grid Index")
plt.ylabel("TotalElectrostaticPotential (eV)")
plt.savefig("dspawpy_proj/dspawpy_tests/outputs/us/3pot_ave.png", dpi=300)  # Image name
```

Processing the electrostatic potential file obtained from Section 3.3 of *Application Cases* yields the following vacuum direction potential function curve:




### API: write_VESTA(), write_delta_rho_vesta(), average_along_axis()

- The `write_VESTA` function handles the visualization of volumetric data:

`dspawpy.io.write.`**`write_VESTA`**(*in_filename: str*, *data_type: str*, *out_filename: str = 'DS-PAW.cube'*, *subtype: str | None = None*, *format: str | None = 'cube'*, *compact: bool = False*, *inorm: bool = False*, *gridsize: Sequence[int] | None = None*)

Read data from a json or h5 file containing electronic system information and write to a VESTA formatted file.

**Parameters**

- **`in_filename`** – Path to a json or h5 file containing electronic system information
- **`data_type`** – Data type, supported values are rho, potential, elf, pcharge, rhoBound
- **`out_filename`** – Output file path, default DS-PAW.cube
- **`subtype`** – Used to specify the subtype of data_type, default is None, which will read the TotalElectrostaticPotential data of potential
- **`format`** – Output data format, supports cube and vesta (vasp), default is cube, case-insensitive
- **`compact`** – Each data point for each grid is placed on a new line, reducing the file size by decreasing the number of spaces (this does not affect the parsing of VESTA software), default is False
- **`inorm`** – Whether to normalize the volume data so that the sum is 1, default is False

– **gridsize** – The redefined number of grid points, in the format (ngx, ngy, ngz), default is None, which uses the original number of grid points

**Returns**
VESTA formatted file

**Return type**
out_filename

### Examples

```
>>> from dspawpy.io.write import write_VESTA
>>> write_VESTA("dspawpy_proj/dspawpy_tests/inputs/2.2/rho.json", "rho", out_
↪filename='dspawpy_proj/dspawpy_tests/outputs/doctest/rho.cube')
==> ...rho.cube...
```

```
>>> from dspawpy.io.write import write_VESTA
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.7/potential.h5",
...     data_type="potential",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/my_potential.
↪cube",
...     subtype='TotalElectrostaticPotential', # or 'TotalLocalPotential'
...     gridsize=(50,50,50), # all integer, can be larger or less than the␣
↪original gridsize
... )
Interpolating volumetric data...
volumetric data interpolated
==> ...my_potential.cube...
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.8/elf.h5",
...     data_type="elf",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/elf.cube",
... )
==> ...elf.cube...
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.9/pcharge.h5",
...     data_type="pcharge",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/pcharge.cube",
... )
==> ...pcharge.cube...
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.7/potential.h5",
...     data_type="potential",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/my_potential.
↪vasp",
...     subtype='TotalElectrostaticPotential', # or 'TotalLocalPotential'
...     gridsize=(50,50,50), # all integer, can be larger or less than the␣
↪original gridsize
... )
Interpolating volumetric data...
volumetric data interpolated
==> ...my_potential.vasp...
>>> write_VESTA(
```

(continues on next page)

(continued from previous page)

```
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.8/elf.h5",
...     data_type="elf",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/elf.vasp",
... )
==> ...elf.vasp...
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.9/pcharge.h5",
...     data_type="pcharge",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/pcharge.vasp",
... )
==> ...pcharge.vasp...
```

```
>>> write_VESTA(
...     in_filename="dspawpy_proj/dspawpy_tests/inputs/2.7/potential.h5",
...     data_type="potential",
...     out_filename="dspawpy_proj/dspawpy_tests/outputs/doctest/my_potential.
↪txt",
...     subtype='TotalElectrostaticPotential', # or 'TotalLocalPotential'
...     gridsize=(50,50,50), # all integer, can be larger or less than the␣
↪original gridsize
... )
Interpolating volumetric data...
volumetric data interpolated
==> ...my_potential.txt...
>>> with open("dspawpy_proj/dspawpy_tests/outputs/doctest/my_potential.txt") as␣
↪t:
...     contents = t.readlines()
...     for line in contents[:10]:
...         print(line.strip())
# 2 atoms
# 50 50 50 grid size
# x y z value
0.000 0.000 0.000      0.3279418
0.055 0.055 0.000     -0.0740864
0.110 0.110 0.000     -0.8811763
0.165 0.165 0.000     -2.1283865
0.220 0.220 0.000     -4.0559145
0.275 0.275 0.000     -6.8291030
0.330 0.330 0.000    -10.1550909
```

volumetricData refers to physical quantities that vary with spatial position, such as charge density rho, potential energy function potential, localized charge density elf, partial charge density pcharge, and solvent-bound charge density rhoBound. This data is stored in the volumetricData type in DS-PAW.

- The `write_delta_rho_vesta` function is responsible for handling the visualization of differential volumetricData:

  `dspawpy.io.write.`**`write_delta_rho_vesta`**(*total: str, individuals: list[str], output: str = 'delta_rho.cube', format: str = 'cube', compact: bool = False, inorm: bool = False, gridsize: Sequence | None = None, data_type: str | None = 'rho', subtype: str | None = None*)

  Charge density differential visualization

DeviceStudio does not currently support large files; it is temporarily written in a format that can be opened with VESTA.

**Parameters**

- **total** – Path to the total charge density file of the system, can be in h5 or json format
- **individuals** – Paths to the charge density files of each component in the system, can be in h5 or json format
- **output** – Output file path, default delta_rho.cube
- **format** – Output data format, supports cube and vasp, default to cube
- **compact** – Each data point for each grid is placed on a new line, and the file size is reduced by reducing the number of spaces (this does not affect the parsing by VESTA software), default is False
- **inorm** – Whether to normalize the volume data so that the sum is 1, default is False
- **gridsize** – Redefined grid number, format as (ngx, ngy, ngz), default is None, use the original grid number

**Returns**
A charge density file after the difference of charges (total - individual1 - individual2 - )

**Return type**
output

### Examples

```
>>> from dspawpy.io.write import write_delta_rho_vesta
>>> write_delta_rho_vesta(total='dspawpy_proj/dspawpy_tests/inputs/supplement/
↪AB.h5',
...     individuals=['dspawpy_proj/dspawpy_tests/inputs/supplement/A.h5',
↪'dspawpy_proj/dspawpy_tests/inputs/supplement/B.h5'],
...     output='dspawpy_proj/dspawpy_tests/outputs/doctest/delta_rho.cube')
==> ...delta_rho.cube...
```

- The `average_along_axis` function handles averaging volumetric data along a specific axis:

`dspawpy.plot.`**`average_along_axis`**(*datafile: str = 'potential.h5', task: str = 'potential', axis: int = 2, smooth: bool = False, smooth_frac: float = 0.8, raw: bool = False, subtype: str | None = None, verbose: bool = False, **kwargs*)

Plot the average curve of a physical quantity along a certain axis

**Parameters**

- **datafile** – Path to an h5 or json file, or a folder containing any of these files, default potential.h5
- **task** – Task type, can be rho, potential, elf, pcharge, rhoBound
- **axis** – Along which axis to plot the potential curve, default is 2
- **smooth** – Whether to smooth, default False
- **smooth_frac** – Smoothing coefficient, default 0.8
- **raw** – Whether to return plot data to a CSV file
- **subtype** – Used to specify the task data subtype, default None, representing drawing Potential/TotalElectrostaticPotential

– ****kwargs** – Other parameters, passed to matplotlib.pyplot.plot

**Returns**
Can be passed to other functions for further processing

**Return type**
axes

### Examples

```
>>> from dspawpy.plot import average_along_axis
```

Read data from the potential.h5 file, plot, and save the original plot data to a CSV file

```
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/3.3/
↪rho.h5', task='rho', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/rho_h5.png')
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/3.3/
↪rho.json', task='rho', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/rho_json.png')
```

```
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.7/
↪potential.h5', task='potential', axis=2, smooth=True, smooth_frac=0.8,␣
↪raw=True)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/potential_h5.png')
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.7/
↪potential.json', task='potential', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/potential_json.png')
```

```
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.8/
↪elf.h5', task='elf', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/elf_h5.png')
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.8/
↪elf.json', task='elf', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/elf_json.png')
```

```
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.9/
↪pcharge.h5', task='pcharge', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/pcharge_h5.png')
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.9/
↪pcharge.json', task='pcharge', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/pcharge_json.png')
```

```
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.28/
↪rhoBound.h5', task='rhoBound', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/rhoBound_h5.png')
>>> plt = average_along_axis(datafile='dspawpy_proj/dspawpy_tests/inputs/2.28/
↪rhoBound.json', task='rhoBound', axis=2, smooth=True, smooth_frac=0.8)
>>> plt.savefig('dspawpy_proj/dspawpy_tests/outputs/doctest/rhoBound_json.png')
```

**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, its possible that the program you are using (e.g., MobaXterm) is incompatible with the QT libraries. You should either switch programs (e.g., VSCode or the systems built-in terminal command line) or add the following code, starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

## 8.4 band data processing

**Note**

1. The script calls get_band_data() to read data, and setting efermi=XX during data reading can shift the energy zero point to the specified value; setting zero_to_efermi=True can shift the energy zero point to the Fermi level in the read file.
2. When plotting using *pymatgens BSPlotter.get_plot()* in the script, you can set *zero_to_efermi=True* to shift the energy zero to the Fermi level. Due to a critical update in *pymatgen* on August 17, 2023, which changed the return object of the plotting function from *plt* to *axes*, subsequent scripts may become incompatible. Therefore, a conditional statement has been added to handle this in the relevant parts of the users script.
3. For two-step band calculations, obtain the accurate Fermi level from the first-step self-consistent calculation (from the self-consistent *system.json*). If this fails, users can modify the energy zero point when calling *get_band_data* to read data, using the *efermi* parameter. For example: *band_data=get_band_data(band.h5,efermi=-1.5)*
4. When plotting, the script calls *BSPlotter.get_plot* from pymatgen. When the system is determined to be non-metallic, setting *zero_to_efermi* will consider the VBM as the Fermi level energy, rather than the Fermi level from the data file. Therefore, when the system is non-metallic, setting *zero_to_efermi=True* during data reading and setting *zero_to_efermi=True* during plotting will result in different plots.

Running the Python script listed in this section, the program will determine whether the system is metallic. If it is a non-metallic system, you will be prompted to choose whether to shift the Fermi level to the zero energy point; please follow the prompts.

### 8.4.1 Conventional Band Treatment

See `4bandplot.py`:

```
# coding:utf-8
import os
import matplotlib.pyplot as plt
from pymatgen.electronic_structure.plotter import BSPlotter

from dspawpy.io.read import get_band_data

datafile = "dspawpy_proj/dspawpy_tests/inputs/supplement/pband.h5"  # Specifies the data file path
band_data = get_band_data(
    band_dir=datafile,
    syst_dir=None,  # path to system.json file, required only when band_dir is a json file
```

(continues on next page)

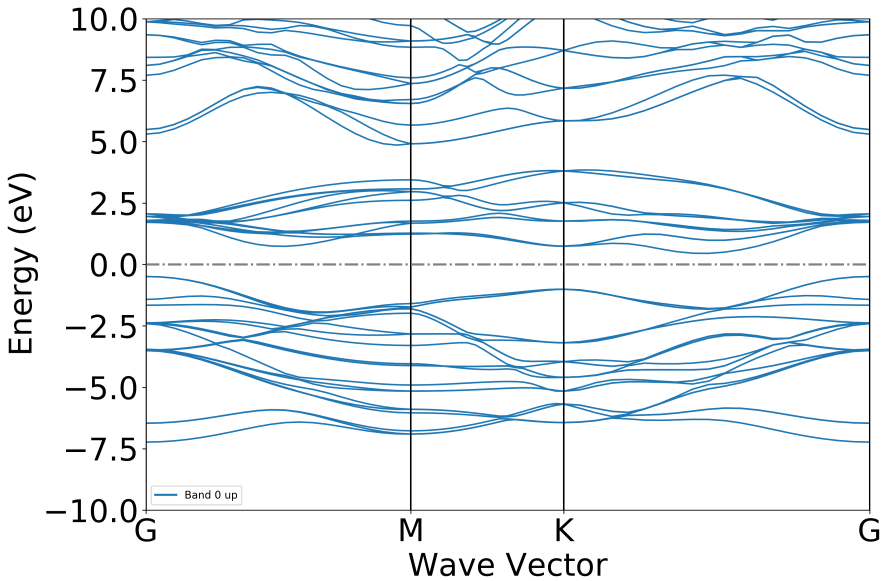
(continued from previous page)

```
    efermi=None,  # Used for manually correcting the Fermi level
    zero_to_efermi=True,  # For non-metallic systems, the zero point energy should be shifted to the Fermi level
)

bsp = BSPlotter(band_data)
axes_or_plt = bsp.get_plot(
    zero_to_efermi=False,  # The data has already been shifted when read, so this should be turned off
    ylim=[-10, 10],  # Range of the y-axis for the band structure plot
    smooth=False,  # Whether to smooth the band structure plot
    vbm_cbm_marker=False,  # Whether to mark the valence band maximum and conduction band minimum in the band structure plot
    smooth_tol=0,  # Threshold for smoothing
    smooth_k=3,  # Order of the smoothing process
    smooth_np=100,  # Number of points for smoothing
)

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

# Add a reference line for the energy zero point
for ax in fig.axes:
    ax.axhline(0, lw=2, ls="-.", color="gray")

figname = "dspawpy_proj/dspawpy_tests/outputs/us/4bandplot.png"  # Filename for the output band plot
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
fig.savefig(figname, dpi=300)
```

**Note**

For band structure calculations, an accurate Fermi level is required, which is obtained from the self-consistent calculation (from system.json). If the acquisition fails, users can modify the efermi parameter in the get_band_data function.

Executing the code will generate a band structure plot similar to the following:




### 8.4.2 The band is projected onto each element separately, with the size of the data points representing the elements contribution to the orbital.

See 4bandplot_elt.py:

```
# coding:utf-8
import os

import matplotlib.pyplot as plt
import numpy as np
from pymatgen.electronic_structure.plotter import BSPlotterProjected

from dspawpy.io.read import get_band_data

datafile = "dspawpy_proj/dspawpy_tests/inputs/supplement/pband.h5"  # Specify the data␣
↪file path
band_data = get_band_data(
    band_dir=datafile,
    syst_dir=None,  # path to system.json file, required only when band_dir is a json␣
↪file
    efermi=None,  # Used to manually adjust the Fermi level
    zero_to_efermi=True,  # For non-metallic systems, shift the zero-point energy to the␣
↪Fermi level
)

bsp = BSPlotterProjected(bs=band_data)  # Initialize the BSPlotterProjected class
axes_or_plt = bsp.get_elt_projected_plots(
    zero_to_efermi=False,  # The data has already been shifted when read, so this should␣
↪be disabled
    ylim=[-8, 5],  # Set the energy range
    vbm_cbm_marker=False,  # Whether to mark the conduction band minimum (CBM) and␣
↪valence band maximum (VBM)
)
```

(continues on next page)

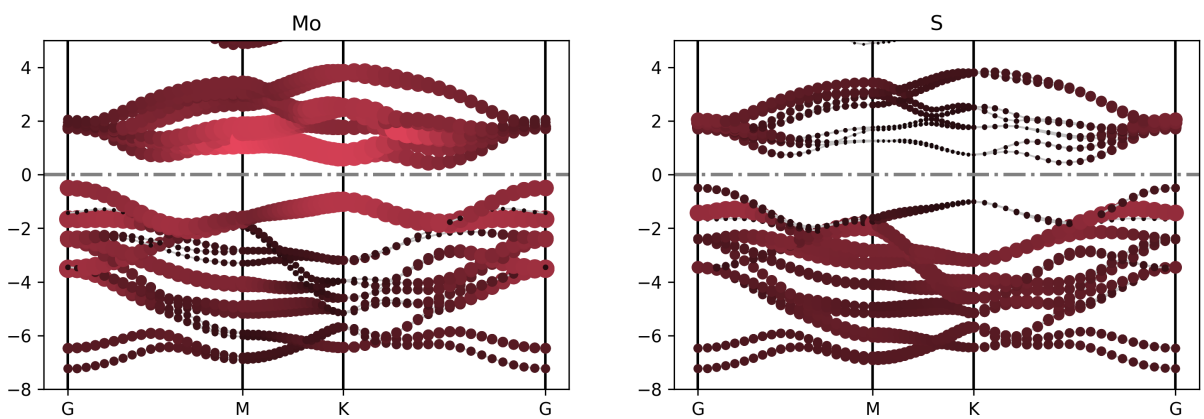
(continued from previous page)

```python

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif np.iterable(axes_or_plt):
    fig = np.asarray(axes_or_plt).flatten()[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

# Add a reference line for the energy zero point
for ax in fig.axes:
    ax.axhline(0, lw=2, ls="-.", color="gray")

figname = "dspawpy_proj/dspawpy_tests/outputs/us/4bandplot_elt.png"  # The filename for␣
↪the output band plot
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
fig.savefig(figname, dpi=300)
```

**Note**

1. To plot projected band structure data, use the BSPlotterProjected module.
2. Use the *get_elt_projected_plots* function in the *BSPlotterProjected* module to plot band diagrams with orbital contributions for each element.

Executing the code will generate band plots similar to the following:




**Warning**

If you execute the script above by connecting to a remote server via SSH, and you encounter QT-related error messages, its possible that the program youre using (such as MobaXterm) is incompatible with the QT libraries. You can either switch to a different program (e.g., VSCode or the systems built-in terminal command line), or add the following code, starting on the second line of your Python script:

```python
import matplotlib
matplotlib.use('agg')
```

### 8.4.3 Band projections onto different elements different orbitals

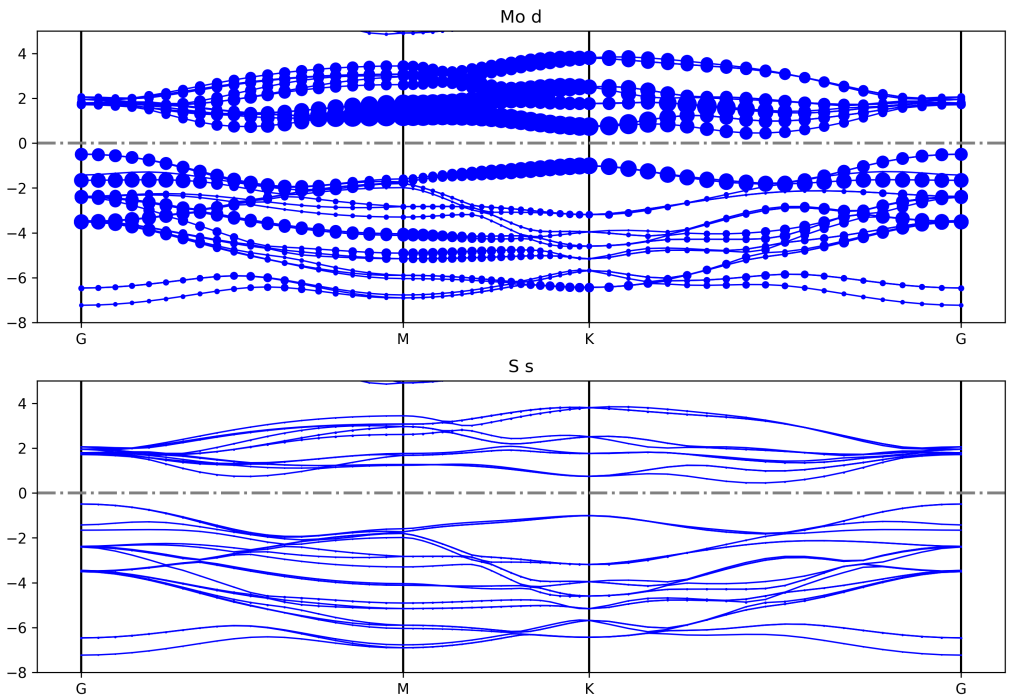
Refer to 4bandplot_elt_orbit.py:

```python
# coding:utf-8
import os

import matplotlib.pyplot as plt
import numpy as np
from pymatgen.electronic_structure.plotter import BSPlotterProjected

from dspawpy.io.read import get_band_data

datafile = "dspawpy_proj/dspawpy_tests/inputs/supplement/pband.h5"  # Specify the data file path
band_data = get_band_data(
    band_dir=datafile,
    syst_dir=None,  # path to system.json file, only required when band_dir is a json file
    efermi=None,  # Used for manually correcting the Fermi level
    zero_to_efermi=True,  # For non-metallic systems, shift the zero point energy to the Fermi level
)

bsp = BSPlotterProjected(bs=band_data)  # Initialize the BSPlotterProjected class
# Select elements and orbitals, create a dictionary
dict_elem_orbit = {"Mo": ["d"], "S": ["s"]}

axes_or_plt = bsp.get_projected_plots_dots(
    dictio=dict_elem_orbit,
    zero_to_efermi=False,  # The data has already been shifted when read, so this should be turned off
    ylim=[-8, 5],  # Set the energy range
    vbm_cbm_marker=False,  # Whether to mark the conduction band minimum and valence band maximum
)

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif np.iterable(axes_or_plt):
    fig = np.asarray(axes_or_plt).flatten()[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

# Add a reference line for the energy zero point
for ax in fig.axes:
    ax.axhline(0, lw=2, ls="-.", color="gray")

figname = "dspawpy_proj/dspawpy_tests/outputs/us/4bandplot_elt_orbit.png"  # Filename for the output band plot
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
fig.savefig(figname, dpi=300)
```

**Note**

1. Use the *get_projected_plots_dots* method in the *BSPlotterProjected* module, which allows users to customize the band structure plots by specifying elements and orbitals (L) to be plotted.
2. For example, get_projected_plots_dots({Mo: [d], S: [s]}) plots the d-orbitals of Mo and the s-orbitals of S.

Executing the code will generate a band structure plot similar to the following:




**Warning**

If you execute the above script by connecting to a remote server via SSH, and QT-related error messages appear, it may be due to incompatibility between the program used (e.g., MobaXterm) and the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.4.4 Projecting band structure onto different atomic orbitals

See `4bandplot_patom_porbit.py`:

```python
# coding:utf-8
import os

import matplotlib.pyplot as plt
import numpy as np
from pymatgen.electronic_structure.plotter import BSPlotterProjected

from dspawpy.io.read import get_band_data

datafile = "dspawpy_proj/dspawpy_tests/inputs/supplement/pband.h5"  # Specify the data file path
band_data = get_band_data(
    band_dir=datafile,
    syst_dir=None,  # path to system.json file, required only when band_dir is a JSON file
    efermi=None,  # Used to manually adjust the Fermi level
    zero_to_efermi=True,  # For non-metallic systems, shift the zero point energy to the Fermi level
)

bsp = BSPlotterProjected(bs=band_data)
# Specify elements, orbitals, and atomic numbers
dict_elem_orbit = {"Mo": ["px", "py", "pz"]}
dict_elem_index = {"Mo": [1]}

axes_or_plt = bsp.get_projected_plots_dots_patom_pmorb(
    dictio=dict_elem_orbit,  # Specify the element-orbit dictionary
    dictpa=dict_elem_index,  # Specify the element-atomic number dictionary
    sum_atoms=None,  # Whether to sum over atoms
    sum_morbs=None,  # Whether to sum orbitals
    zero_to_efermi=False,  # Data has already been shifted during reading, should be turned off here
    ylim=None,  # Set the energy range
    vbm_cbm_marker=False,  # Whether to mark the conduction band minimum and valence band maximum
    selected_branches=None,  # Specify the energy band branches to be plotted
    w_h_size=(12, 8),  # Set image width and height
    num_column=None,  # Number of images displayed per row
)

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif np.iterable(axes_or_plt):
    fig = np.asarray(axes_or_plt).flatten()[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

# Add a reference line for the energy zero point
for ax in fig.axes:
    ax.axhline(0, lw=2, ls="-.", color="gray")

figname = " dspawpy_proj/dspawpy_tests/outputs/us/4band_patom_porbit.png"  # Output bandpass figure filename
```

(continues on next page)

(continued from previous page)

```
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
fig.savefig(figname, dpi=300)
```

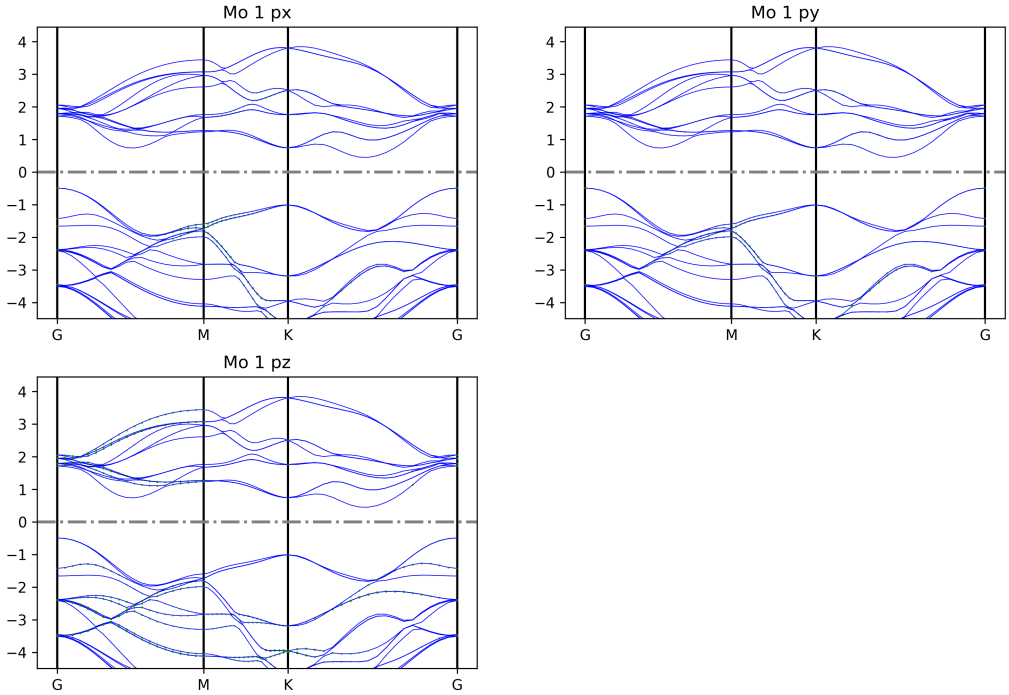
**Note**

1. The *get_projected_plots_dots_patom_pmorb* function in the BSPlotterProjected module offers greater flexibility, allowing users to customize the band diagrams for specific atoms and orbitals.
2. Use *dictpa* to specify the atom, and *dictio* to specify the orbitals of that atom.
3. To superimpose projected components of some atoms or orbitals, specify the *sum_atoms* or *sum_morbs* parameters according to the documentation of the *get_projected_plots_dots_patom_pmorb* function.

**Warning**

1. If only a single orbital is selected and the orbital name has more than one letter (e.g., px, dxy, dxz), the get_projected_plots_dots_patom_pmorb function will raise an error. See here for details.

Executing the code will generate band diagrams similar to the following:



**Warning**

If you execute the above script by connecting to a remote server via SSH, and you encounter QT-related error messages, its possible that the program youre using (e.g., MobaXterm) is incompatible with the QT libraries. Either switch to another program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

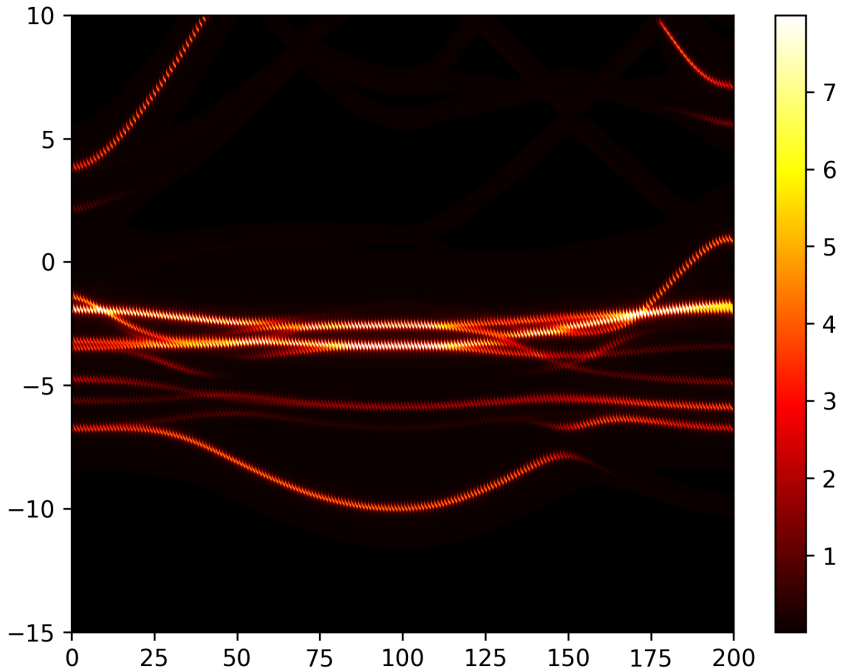
```
import matplotlib
matplotlib.use('agg')
```

### 8.4.5 Band unfolding processing

See `4bandunfolding.py`:

```
# coding:utf-8
import os

from dspawpy.plot import plot_bandunfolding

plt = plot_bandunfolding(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.22.1/band.h5",  # Read data
    ef=None,  # Fermi level, read from the file
    de=0.05,  # Band width, default 0.05
    dele=0.06,  # Band gap, default 0.06
)

plt.ylim(-15, 10)
figname = "dspawpy_proj/dspawpy_tests/outputs/us/4bandunfolding.png"  # Output band␣
↪structure plot filename
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
plt.savefig(figname, dpi=300)
# plt.show()
```

Executing the code yields a band diagram similar to the following:




**Warning**

**Warning**

This feature currently does not support setting the Fermi level of non-metallic materials as the zero-energy point (the default is the valence band top as the zero-energy point).

**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, it might be due to incompatibility between the program used (such as MobaXterm, etc.) and the QT library. You can either switch programs (e.g., VSCode or the systems built-in terminal) or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

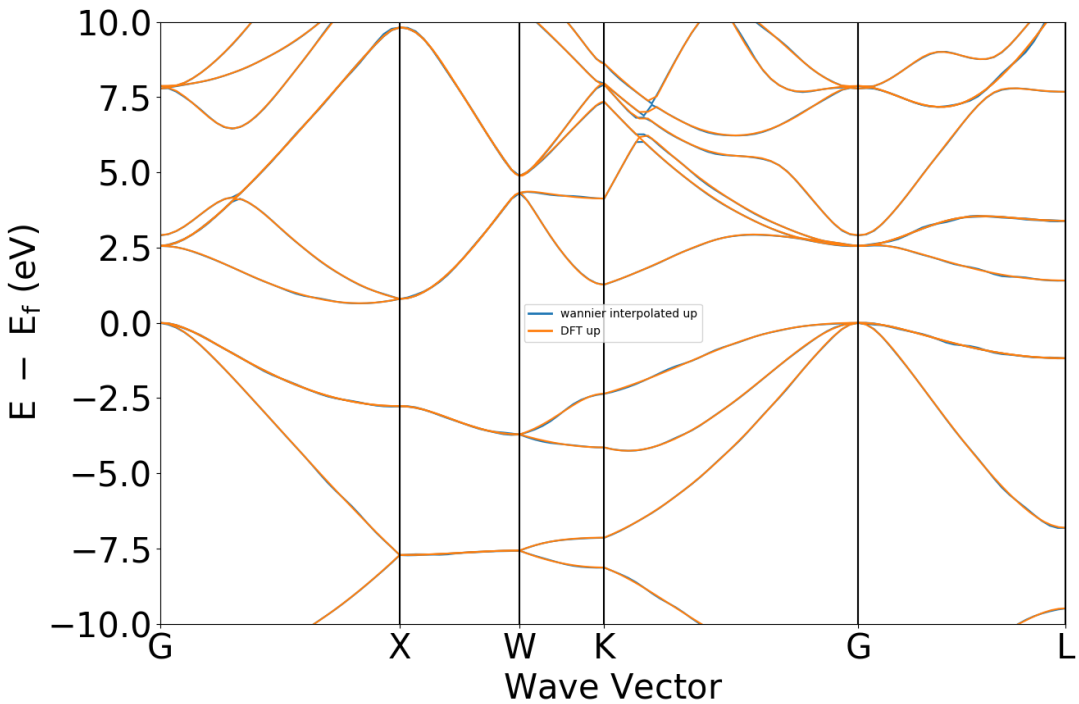
### 8.4.6 band-compare band structure comparison figure processing

Plotting regular band structure and Wannier band structure on the same figure.

Refer to `4bandcompare.py`:

```python
# coding:utf-8
import os

from pymatgen.electronic_structure.plotter import BSPlotter

from dspawpy.io.read import get_band_data

band_data = get_band_data(
    band_dir="dspawpy_proj/dspawpy_tests/inputs/2.30/wannier.h5",  # Wannier band file path
    syst_dir=None,  # system.json file path, only needed when band_dir is a json file
    efermi=None,  # Used for manually adjusting the Fermi level
    zero_to_efermi=False,  # Whether to shift zero energy to the Fermi level
)
bsp = BSPlotter(bs=band_data)
band_data = get_band_data(
    band_dir="dspawpy_proj/dspawpy_tests/inputs/2.3/band.h5",  # Read DFT band structure
    syst_dir=None,  # path to system.json file, required only when band_dir is a json file
    efermi=None,  # Used for manually correcting the Fermi level
    zero_to_efermi=False,  # Whether to shift the zero point energy to the Fermi level
)

bsp2 = BSPlotter(bs=band_data)
bsp.add_bs(bsp2._bs)
axes_or_plt = bsp.get_plot(
    zero_to_efermi=True,  # Move the zero energy level to the Fermi level
    ylim=[-10, 10],  # Energy band plot y-axis range
    smooth=False,  # Whether to smooth the band structure plot
    vbm_cbm_marker=False,  # Whether to mark the valence band maximum and conduction band minimum in the band structure plot
    smooth_tol=0,  # Threshold for smoothing
    smooth_k=3,  # Order of the smoothing process
    smooth_np=100,  # Number of points for smoothing
    bs_labels=["wannier interpolated", "DFT"],  # Band structure labels
)

import matplotlib.pyplot as plt  # noqa: E402

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

# Add a reference line for the energy zero point
for ax in fig.axes:
    ax.axhline(0, lw=2, ls="-.", color="gray")

figname = "dspawpy_proj/dspawpy_tests/outputs/us/4wanierBand.png"  # File name for the output band structure plot
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)
fig.savefig(figname, dpi=300)
```

Executing the code will generate band comparison curves similar to the following:




### API: get_band_data()

- The `get_band_data` function is responsible for reading band structure data as follows:

`dspawpy.io.read.`**`get_band_data`**(*band_dir: str*, *syst_dir: str | None = None*, *efermi: float | None = None*, *zero_to_efermi: bool = False*, *verbose: bool = False*) → BandStructureSymmLine

Reads band structure data from an h5 or json file and constructs a BandStructureSymmLine object.

**Parameters**

- **band_dir** –
  - Path to the band structure file, band.h5 / band.json, or a directory containing band.h5 / band.json
  - Note that wannier.h5 can also be read using this function, but band_dir does not support folder types
- **syst_dir** – Path to system.json, prepared only for auxiliary processing of Wannier data (structure and Fermi level are read from it)
- **efermi** – Fermi level, if the Fermi level in the h5 file is incorrect, it can be specified using this parameter
- **zero_to_efermi** – Whether to shift the Fermi level to 0

**Return type**

BandStructureSymmLine

### Examples

```
>>> from dspawpy.io.read import get_band_data
>>> band = get_band_data(band_dir='dspawpy_proj/dspawpy_tests/inputs/2.3/band.h5
↪')
>>> band = get_band_data(band_dir='dspawpy_proj/dspawpy_tests/inputs/2.4/band.h5
↪')
>>> band = get_band_data(band_dir='dspawpy_proj/dspawpy_tests/inputs/2.4/band.
↪json')
```

If you want to process Wannier band structures by specifying wannier.json, you need to additionally specify the syst_dir parameter.

```
>>> band = get_band_data(band_dir='dspawpy_proj/dspawpy_tests/inputs/2.30/
↪wannier.h5')
>>> band = get_band_data(band_dir='dspawpy_proj/dspawpy_tests/inputs/2.30/
↪wannier.json', syst_dir='dspawpy_proj/dspawpy_tests/inputs/2.30/system.json')
```

**Warning**

If you are running the above script by connecting to a remote server via SSH and encounter QT-related error messages, it may be due to incompatibility between the program you are using (such as MobaXterm, etc.) and the QT libraries. You can either switch to another program (such as VSCode or the systems built-in terminal), or add the following code, starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

## 8.5 DOS Data Processing

### 8.5.1 Total Density of States

See `5dosplot_total.py`:

```
# coding:utf-8
import os

from pymatgen.electronic_structure.plotter import DosPlotter

from dspawpy.io.read import get_dos_data
from dspawpy.plot import plot_dos

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/3.2.4/dos.h5",  # Read projected density␣
↪of states data
    return_dos=False,  # If False, always return a CompleteDos object (regardless of␣
↪whether projection was enabled during calculation)
)
dos_plotter = DosPlotter(
    zero_at_efermi=True,  # Whether to set the Fermi level as the zero point
    stack=False,  # True indicates drawing an area chart
    sigma=None,  # Gaussian broadening, None indicates no smoothing process
```

(continues on next page)

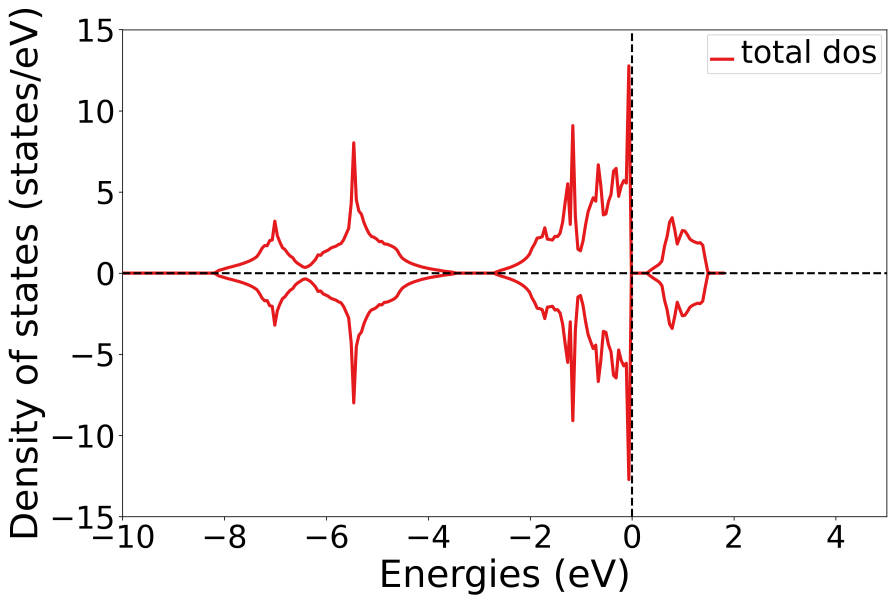
(continued from previous page)

```
)
dos_plotter.add_dos(
    label="total dos", dos=dos_data
)  # Set the legend for the density of states plot  # Pass the density of states data

ax = plot_dos(
    dosplotter=dos_plotter,
    xlim=[-10, 5],  # Set the energy range
    ylim=[-15, 15],  # Set the density of states range
)
ax.axhline(0, lw=2, ls="-.", color="gray")

filename = "dspawpy_proj/dspawpy_tests/outputs/us/5dos_total.png"  # File name for the␣
↪output density of states plot
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)

fig = ax.get_figure()
fig.savefig(filename, dpi=300)
```

**Note**

1. Use the *get_dos_data* function to convert the *dos.h5* file obtained from DS-PAW calculations into a format supported by pymatgen.
2. Use the DosPlotter module to obtain the data from the DS-PAW calculated dos.h5 file.
3. The DosPlotter function can pass parameters: the *stack* parameter indicates whether to fill the DOS plots, and *zero_at_efermi* indicates whether to set the Fermi energy to zero in the DOS plot. Here, *stack=False* and *zero_at_efermi=False* are set.
4. Use *add_dos* in the *DosPlotter* module to add the DOS data.
5. Use the *get_plot* function in the *DosPlotter* module to plot the DOS.

Executing the code will generate a density of states plot similar to the following:




**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, its likely that the program youre using (such as MobaXterm, etc.) is incompatible with the QT libraries. You can either switch programs (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.5.2 Project Density of States onto different orbitals

See `5dosplot_spd.py`:

```
# coding:utf-8
import os

from pymatgen.electronic_structure.plotter import DosPlotter

from dspawpy.io.read import get_dos_data
from dspawpy.plot import plot_dos

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/3.2.4/dos.h5",  # Read projected DOS data
    return_dos=False,  # If False, always return a CompleteDos object (regardless of whether projection was enabled during calculation)
)
dos_plotter = DosPlotter(
    zero_at_efermi=True,  # Whether to set the Fermi level as the zero point
    stack=False,  # True indicates drawing an area chart
    sigma=None,  # Gaussian broadening, None indicates no smoothing is applied
)
```

(continues on next page)

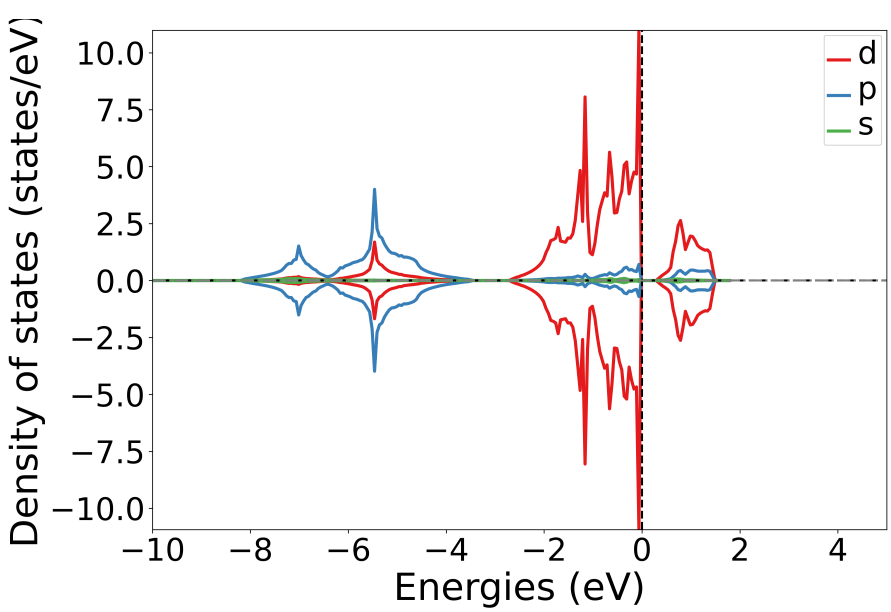
(continued from previous page)

```
dos_plotter.add_dos_dict(
    dos_dict=dos_data.get_spd_dos(),
    key_sort_func=None,  # Orbital projection  # Specifies the sorting function
)
ax = plot_dos(
    dosplotter=dos_plotter,
    xlim=[-10, 5],  # Set the energy range
    ylim=None,  # Set the density of states range
)

ax.axhline(0, lw=2, ls="-.", color="gray")

filename = "dspawpy_proj/dspawpy_tests/outputs/us/5dos_spd.png"  # Filename of the␣
↪output density of states plot
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)

fig = ax.get_figure()
fig.savefig(filename, dpi=300)
```

**Note**

Use the *add_dos_dict* function in the *DosPlotter* module to obtain the projected density of states (DOS) data, and then use *get_spd_dos* to project the information onto spd orbitals.

The code execution will produce a density of states plot similar to the following:




**Warning**

If you encounter QT-related error messages when executing the above script via SSH connection to a remote

server, it might be due to incompatibility between the program used (e.g., MobaXterm) and the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.5.3 Projecting the density of states onto different elements

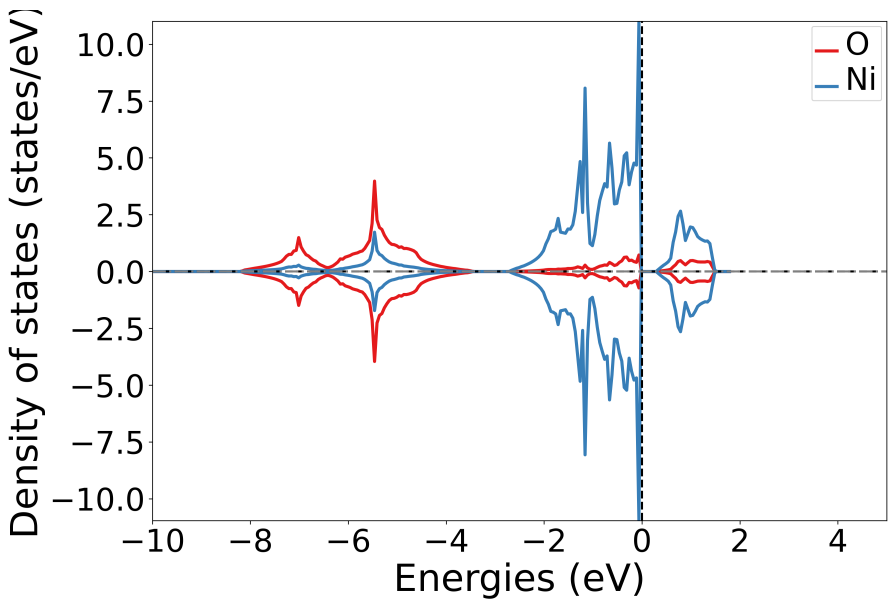
See also `5dosplot_elt.py`:

```
# coding:utf-8
import os

from pymatgen.electronic_structure.plotter import DosPlotter

from dspawpy.io.read import get_dos_data
from dspawpy.plot import plot_dos

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/3.2.4/dos.h5",  # Reads projected DOS data
    return_dos=False,  # If False, always returns a CompleteDos object (regardless of␣
↪whether projection was enabled during calculation)
)
dos_plotter = DosPlotter(
    zero_at_efermi=True,  # Whether to set the Fermi level as the zero point
    stack=False,  # True indicates drawing an area chart
    sigma=None,  # Gaussian broadening, None indicates no smoothing is applied
)
dos_plotter.add_dos_dict(
    dos_dict=dos_data.get_element_dos(),
    key_sort_func=None,  # Projected DOS for elements  # Specify the sorting function
)

ax = plot_dos(
    dosplotter=dos_plotter,
    xlim=[-10, 5],  # Set the energy range
    ylim=None,  # Set the density of states range
)
ax.axhline(0, lw=2, ls="-.", color="gray")

filename = "dspawpy_proj/dspawpy_tests/outputs/us/5dos_elt.png"  # Filename for the␣
↪output density of states plot
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)

fig = ax.get_figure()
fig.savefig(filename, dpi=300)
```

**Note**

Use the *add_dos_dict* function in the DosPlotter module to obtain projected density of states data, then use *get_element_dos* to output the projected information according to different elements.

The code execution will produce a density of states plot similar to the following:




**Warning**

If you encounter QT-related error messages when executing the above script via SSH connection to a remote server, it might be due to incompatibility between the program used (e.g., MobaXterm) and the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.5.4 Projecting the density of states onto different orbitals of different atoms

See `5dosplot_atom_orbit.py`:

```
# coding:utf-8
import os

from pymatgen.electronic_structure.core import Orbital
from pymatgen.electronic_structure.plotter import DosPlotter

from dspawpy.io.read import get_dos_data
from dspawpy.plot import plot_dos

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/3.2.4/dos.h5",  # Reads projected density␣
↪of states data
    return_dos=False,  # If False, always return a CompleteDos object (regardless of␣
↪whether projection was enabled during calculation)
)
dos_plotter = DosPlotter(
```

(continues on next page)

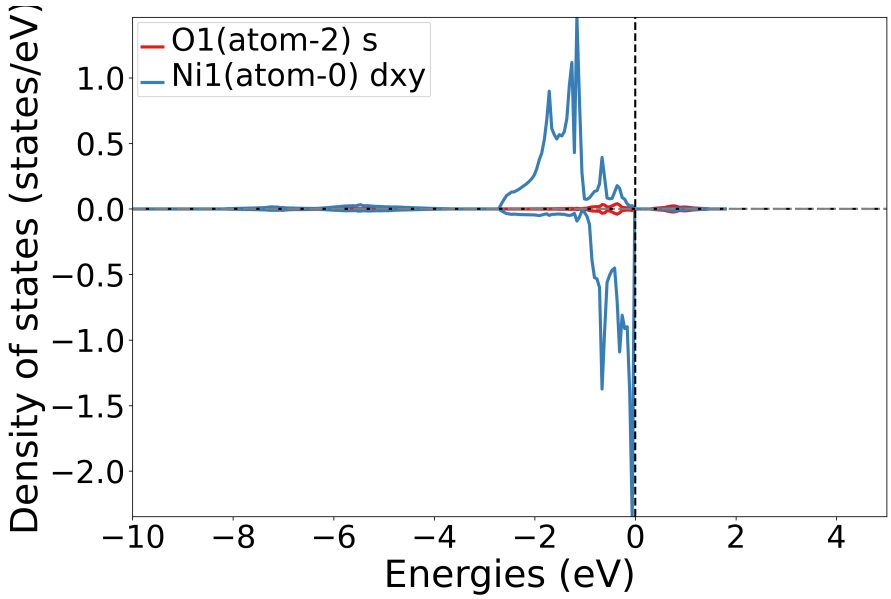
(continued from previous page)

```
    zero_at_efermi=True,  # Whether to set the Fermi level as the zero point
    stack=False,  # True indicates drawing an area plot
    sigma=None,  # Gaussian broadening, None indicates no smoothing treatment
)

# ! Specify atomic number and orbital
dict_index_orbit = {0: ["dxy"], 2: ["s"]}

print("Plotting...")
for index in dict_index_orbit:
    _os = dict_index_orbit[index]
    _e = str(dos_data.structure.sites[index].species)
    for _orb in _os:
        dos_plotter.add_dos(
            f"{_e}(atom-{index}) {_orb}",  # label
            dos_data.get_site_orbital_dos(
                dos_data.structure[index],
                getattr(Orbital, _orb),
            ),
        )

ax = plot_dos(
    dosplotter=dos_plotter,
    xlim=[-10, 5],  # Set the energy range
    ylim=None,  # Set the density of states range
)
ax.axhline(0, lw=2, ls="-.", color="gray")

figname = "dspawpy_proj/dspawpy_tests/outputs/us/5dos_atom_orbit.png"  # Output density␣
↪of states figure filename
os.makedirs(os.path.dirname(os.path.abspath(figname)), exist_ok=True)

fig = ax.get_figure()
fig.savefig(figname, dpi=300)
```

**Note**

1. Use the *get_site_orbital_dos* function to extract the contribution of a specific atom and specific orbital from the DOS data. *dos_data.structure[0], Orbital(4)* represents obtaining the density of states for the dxy orbital of the first atom; the index in the *get_site_orbital_dos* function starts from 0.
2. Running this script and selecting the element and orbital as prompted will generate the corresponding density of states (DOS) plot.

Executing the code will produce a density of states plot similar to the following:




**Warning**

If you encounter QT-related error messages when executing the above script via SSH connection to a remote server, its likely due to incompatibility between the program youre using (e.g., MobaXterm) and the QT library. Either switch to a different program (such as VSCode or the systems built-in terminal command line), or add the following code to your Python script starting from the second line:

```
import matplotlib
matplotlib.use('agg')
```

### 8.5.5 Projecting the density of states onto the split d-orbitals (t2g, eg) of different atoms

See also `5dosplot_t2g_eg.py`:

```
# coding:utf-8
import os

from pymatgen.electronic_structure.plotter import DosPlotter

from dspawpy.io.read import get_dos_data
from dspawpy.plot import plot_dos

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/3.2.4/dos.h5",  # Read projected density␣
↪of states data
    return_dos=False,  # If False, always returns a CompleteDos object (regardless of␣
↪whether projection was enabled during calculation)
)
dos_plotter = DosPlotter(
    zero_at_efermi=True,  # Whether to set the Fermi level as the zero point
```

(continues on next page)

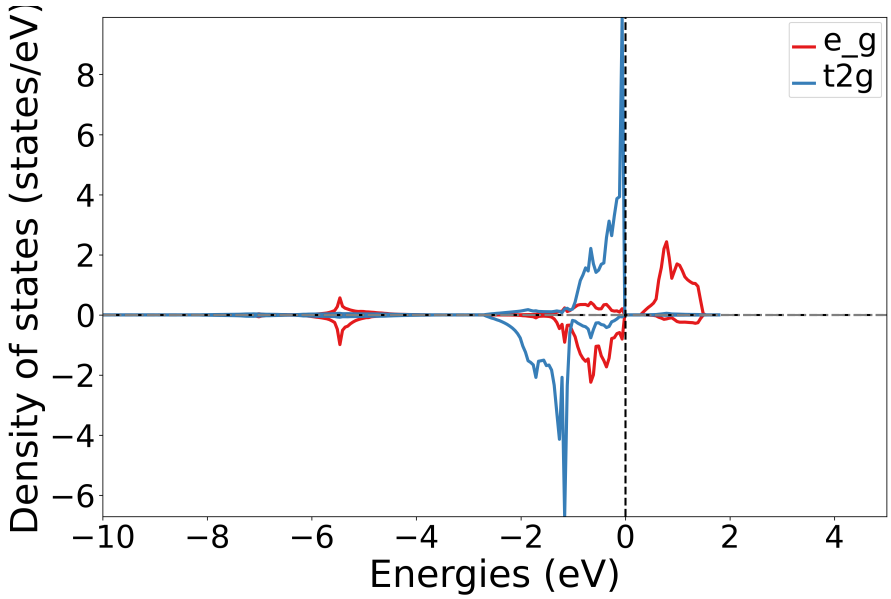
(continued from previous page)

```
    stack=False,  # True indicates drawing an area chart
    sigma=None,  # Gaussian broadening, None indicates no smoothing is applied
)
# print(dos_data.structure)

# Specify the atomic number, starting from 0
ais = [1]

print("Plotting...")
atom_indices = [int(ai) for ai in ais]
for atom_index in atom_indices:
    dos_plotter.add_dos_dict(
        dos_data.get_site_t2g_eg_resolved_dos(dos_data.structure[atom_index]),
    )

ax = plot_dos(
    dosplotter=dos_plotter,
    xlim=[-10, 5],  # Set the energy range
    ylim=None,  # Set the density of states range
)
ax.axhline(0, lw=2, ls="-.", color="gray")

filename = "dspawpy_proj/dspawpy_tests/outputs/us/5dos_t2g_eg.png"  # Output density of␣
↪states plot filename
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)

fig = ax.get_figure()
fig.savefig(filename, dpi=300)
```

**Note**

1. Use the *get_site_t2g_eg_resolved_dos* function to extract the t2g and eg orbital contributions for a specific atom from the DOS data. This retrieves the t2g and eg orbital contributions for the second atom.
2. Running this script and selecting an atom number as prompted will generate the corresponding density of states plot.

Executing the code will generate a density of states plot similar to the following:




**Note**

If the element does not contain d orbitals, a blank image will be drawn.

**Warning**

If you execute the script above by connecting to a remote server via SSH and encounter QT-related error messages, its likely that the program you are using (such as MobaXterm) is incompatible with the QT library. You can either switch programs (e.g., VSCode or the systems built-in terminal command line) or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.5.6 d-centered analysis

Taking the Pb-slab system as an example, a d-band center analysis is performed on Pt atoms:

See `5center_dband.py`:

```
# coding:utf-8
from dspawpy.io.read import get_dos_data
from dspawpy.io.utils import d_band

dos_data = get_dos_data(
    dos_dir="dspawpy_proj/dspawpy_tests/inputs/supplement/dos.h5",  # Read projected density of states data
    return_dos=False,  # If False, always returns a CompleteDos object (regardless of whether projection was enabled during calculation)
)
for spin in dos_data.densities:
```

(continues on next page)

(continued from previous page)

```
    print("spin=", spin)
    c = d_band(spin, dos_data)
    print(c)
```

Executing the code yields results similar to the following:

```
spin=1
-1.785319344084034
```

> **Note**
>
> Currently, only the d-orbital center averaged over all atoms is supported. Element-resolved, atom-projected, or other orbitals are not supported, nor is the selection of spin direction or energy range.

The `get_dos_data` function is responsible for processing density of states data:

### API: get_dos_data()

`dspawpy.io.read.`**`get_dos_data`**(*dos_dir: str*, *return_dos: bool = False*, *verbose: bool = False*)

Read density of states (DOS) data from an h5 or json file, and construct a CompleteDos or DOS object

**Parameters**

- **dos_dir** – Path to the density of states file, dos.h5 / dos.json, or a folder containing dos.h5 / dos.json
- **return_dos** (*bool, optional*) – Whether to return the DOS object. If False, a CompleteDos object is returned uniformly (regardless of whether projection was enabled during calculation)

**Return type**

CompleteDos or Dos

### Examples

```
>>> from dspawpy.io.read import get_dos_data
>>> dos = get_dos_data(dos_dir='dspawpy_proj/dspawpy_tests/inputs/2.5/dos.h5')
>>> dos = get_dos_data(dos_dir='dspawpy_proj/dspawpy_tests/inputs/2.5/dos.h5',␣
↪return_dos=True)
```

## 8.6 bandDos: Displaying Band Structure and Density of States Together

Using the Si system from the application tutorial as an example:

### 8.6.1 Display band structure and density of states in a single figure.

See `6bandDosplot.py`:

```
# coding:utf-8
import os
```

(continues on next page)

(continued from previous page)

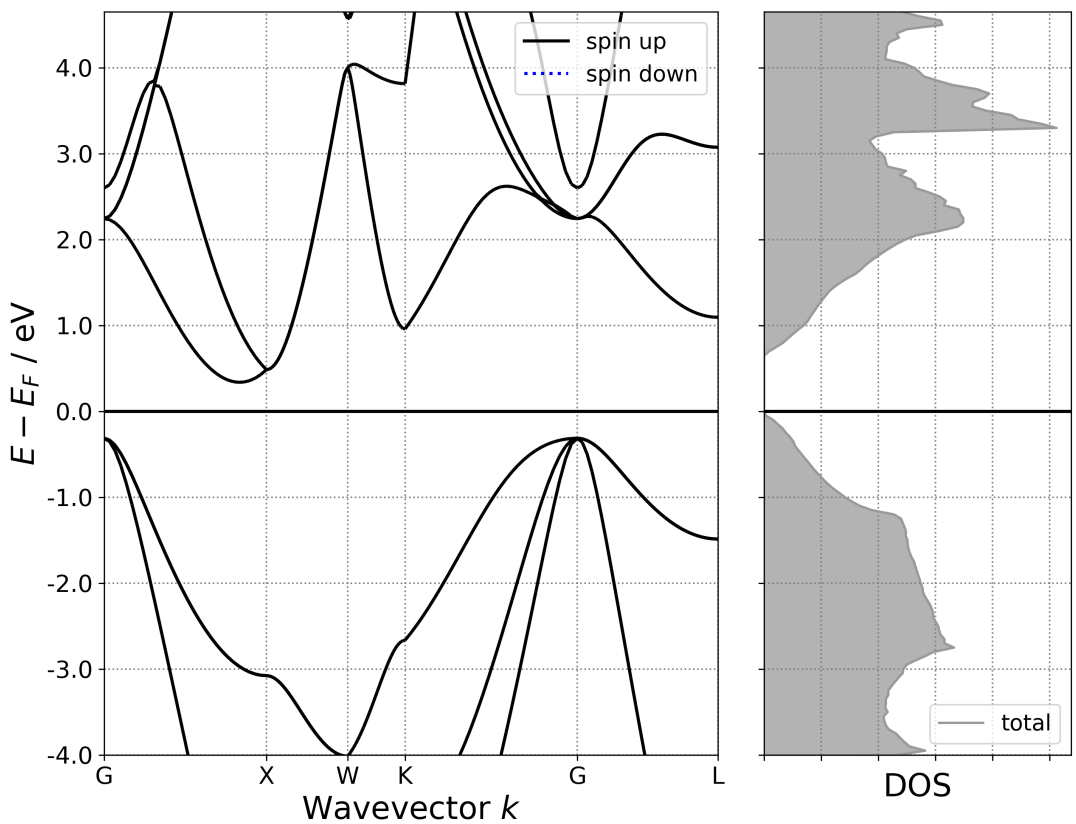
```python

import numpy as np
from matplotlib.axes import Axes
from pymatgen.electronic_structure.plotter import BSDOSPlotter

from dspawpy.io.read import get_band_data, get_dos_data

bandfile = "dspawpy_proj/dspawpy_tests/inputs/2.3/band.h5"  # Normal band data
band_data = get_band_data(
    band_dir=bandfile,
    syst_dir=None,  # path to system.json file, required only when band_dir is a json file
    efermi=None,  # Used for manually correcting the Fermi level
)
band_efermi = band_data.efermi
dosfile = "dspawpy_proj/dspawpy_tests/inputs/2.5/dos.h5"  # Density of states data
dos_data = get_dos_data(
    dos_dir=dosfile,
    return_dos=False,  # If False, always return a CompleteDos object (regardless of whether projection was enabled during calculation)
)
dos_efermi = dos_data.efermi
bdp = BSDOSPlotter(
    bs_projection=None,  # Band structure projection method, None means no projection
    dos_projection=None,  # Projection method for density of states, None means no projection
    vb_energy_range=4,  # Valence band energy range
    cb_energy_range=4,  # Conduction band energy range
    fixed_cb_energy=False,  # Whether to fix the conduction band energy range
    egrid_interval=1,  # Energy grid interval
    font="DejaVu Sans",  # Default is Times New Roman, change to DejaVu Sans to avoid warnings due to missing font on Linux
    axis_fontsize=20,  # Axis font size
    tick_fontsize=15,  # Tick label font size
    legend_fontsize=14,  # Legend font size
    bs_legend="best",  # Band structure legend position
    dos_legend="best",  # Density of States legend position
    rgb_legend=True,  # Use colored legend
    fig_size=(11, 8.5),  # Figure size
)
if band_efermi != dos_efermi:
    print(f"{band_efermi=:.4f} eV")
    print(f"{dos_efermi=:.4f} eV")
    d_efermi = band_efermi - dos_efermi

    print(
        "! Band and DOS Fermi levels are inconsistent, using DOS Fermi level as reference"
    )
    band_data.bands = {spin: v + d_efermi for spin, v in band_data.bands.items()}

    # ! Band and DOS Fermi levels are inconsistent, using Band level as the reference
```

(continues on next page)

(continued from previous page)

```
    # dos_data.energies -= d_efermi

axes_or_plt = bdp.get_plot(
    bs=band_data, dos=dos_data
)  # Pass band data  # Pass density of states data

if isinstance(axes_or_plt, Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif np.iterable(axes_or_plt):
    fig = np.asarray(axes_or_plt).flatten()[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

filename = "dspawpy_proj/dspawpy_tests/outputs/us/6bandDos.png"  # Filename for the band␣
↪structure - density of states plot output
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
fig.savefig(filename, dpi=300)
print("==> Saved", filename)
```

Executing the code yields a band density of states plot similar to the following:



**Warning**

If you are connecting to a remote server via SSH and running the above script, and you encounter QT-related error messages, its possible that the program you are using (such as MobaXterm) is incompatible with the QT libraries. You should either switch programs (e.g., VSCode or the systems built-in terminal command line) or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.6.2 Display band structure and projected density of states on a single plot.

See `6bandPdosplot.py`:

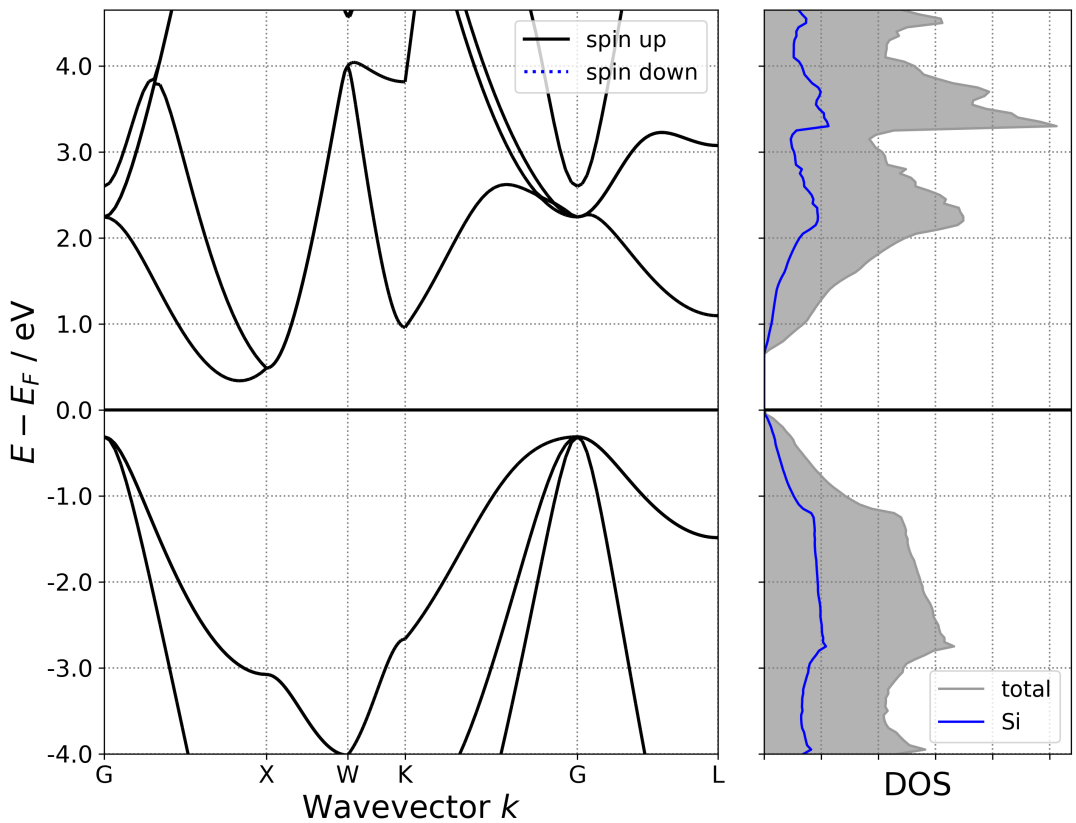
```
# coding:utf-8
import os

import numpy as np
from matplotlib.axes import Axes
from pymatgen.electronic_structure.plotter import BSDOSPlotter

from dspawpy.io.read import get_band_data, get_dos_data

bandfile = "dspawpy_proj/dspawpy_tests/inputs/2.4/band.h5"  # Normal band data
band_data = get_band_data(
    band_dir=bandfile,
    syst_dir=None,  # path to system.json file, required only when band_dir is a json␣
↪file
    efermi=None,  # Used for manually correcting the Fermi level
)
band_efermi = band_data.efermi
dosfile = (
    "dspawpy_proj/dspawpy_tests/inputs/2.6/dos.h5"  # DOS data for projected states
)
dos_data = get_dos_data(
    dos_dir=dosfile,
    return_dos=False,  # If False, always return a CompleteDos object (regardless of␣
↪whether projection was enabled during calculation)
)
dos_efermi = dos_data.efermi
bdp = BSDOSPlotter(
    bs_projection="elements",  # Projection method for band structure, None means no␣
↪projection
    dos_projection="elements",  # Project DOS onto elements
    vb_energy_range=4,  # Valence band energy range
    cb_energy_range=4,  # Conduction band energy range
    fixed_cb_energy=False,  # Whether to fix the conduction band energy range
    egrid_interval=1,  # Energy grid interval
    font="DejaVu Sans",  # Default is Times New Roman, can be changed to DejaVu Sans to␣
↪avoid warnings due to font not being installed on Linux
    axis_fontsize=20,  # Axis font size
    tick_fontsize=15,  # Tick label font size
```

(continues on next page)

(continued from previous page)

```
    legend_fontsize=14,  # Legend font size
    bs_legend="best",  # Band structure legend position
    dos_legend="best",  # Position of the projected density of states legend
    rgb_legend=True,  # Use colored legend
    fig_size=(11, 8.5),  # Figure size
)
if band_efermi != dos_efermi:
    print(f"{band_efermi=:.4f} eV")
    print(f"{dos_efermi=:.4f} eV")
    d_efermi = band_efermi - dos_efermi

    print(
        "! Band and DOS Fermi levels are inconsistent, using DOS Fermi level as reference
↪"
    )
    band_data.bands = {spin: v + d_efermi for spin, v in band_data.bands.items()}

    # ! Band and DOS Fermi levels are inconsistent, using Band level as reference
    # dos_data.energies -= d_efermi

axes_or_plt = bdp.get_plot(
    bs=band_data,
    dos=dos_data,
)  # Pass band structure data  # Pass projected density of states data

if isinstance(axes_or_plt, Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif np.iterable(axes_or_plt):
    fig = np.asarray(axes_or_plt).flatten()[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

filename = "dspawpy_proj/dspawpy_tests/outputs/us/6bandPdos.png"  # filename for the␣
↪band structure-projected density of states plot
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
fig.savefig(filename, dpi=300)
print("==> Saved", filename)
```

Executing the code yields a band-decomposed density of states plot similar to the following:




**Warning**

1. Given projected band data, it will be projected along the element by default; given ordinary band data (or if the system contains more than 4 types of elements), it will not be projected and a warning will be output.
2. Given projected density of states (PDOS) data, projection along elements is also the default. You can switch to projection along orbitals, or no projection at all. For ordinary density of states (DOS) data and without disabling the DOS projection option BSDOSPlotter(dos_projection=None), the pymatgen plotting program will report an error, which is why a 6bandDosplot.py file was specifically prepared, as mentioned above.

**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, its likely due to incompatibility between the program you are using (e.g., MobaXterm) and the QT library. Either switch to a different program (such as VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

## 8.7 optical data processing

Using the *scf.h5* file obtained from a quick start calculation of the optical properties of the Si system as an example (Note: the output file name is the same as the task, task = scf; io.optical = true can calculate optical properties):

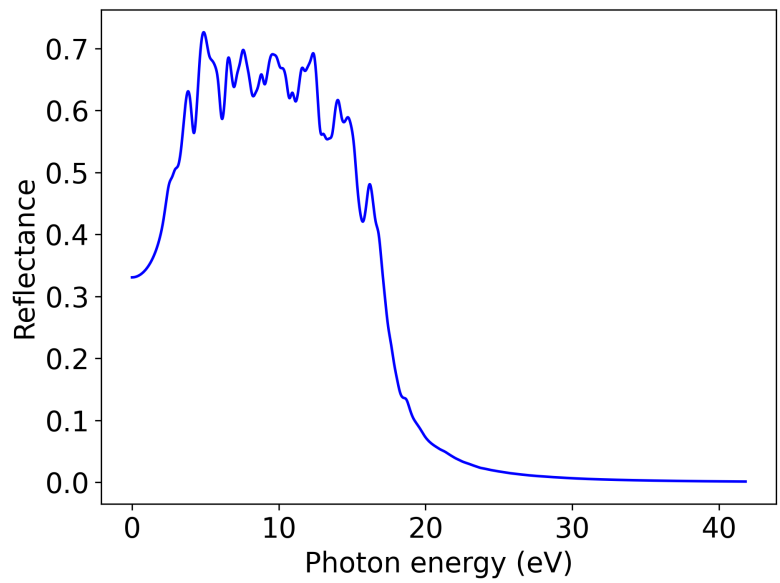
Processing the reflectivity data, referring to `7optical.py`:

```
# coding:utf-8
from dspawpy.plot import plot_optical

plot_optical(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.12/scf.h5",
    keys=["ExtinctionCoefficient", "Reflectance"],
    axes=["X"],  # ["X", "Y", "Z", "XY", "YZ", "ZX"]
    prefix="dspawpy_proj/dspawpy_tests/outputs/optical",  # Where to save, if empty, it␣
↪means the current folder
    save=True,  # Whether to save the image with the tool's name, if False, please refer␣
↪to the script below to save manually
)

# The above function will plot and save the images of ExtinctionCoefficient and␣
↪Reflectance separately
# To plot multiple properties on the same figure, uncomment the following code and set␣
↪the save parameter above to False

# import os
# import matplotlib.pyplot as plt
#
# plt.tick_params(labelsize=16)
# plt.tight_layout()
# filename = "outputs/us/7optical.png"  # Filename for the output optical properties plot
# os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
# plt.savefig(filename, dpi=300)
```

**Note**

Reflectance is an optical property, and users can modify this keyword to AbsorptionCoefficient, ExtinctionCoefficient, or RefractiveIndex based on their needs, corresponding to the absorption coefficient, extinction coefficient, and refractive index, respectively.

Executing the code will generate a curve showing the reflectance as a function of energy, similar to the following:




### API: plot_optical()

`dspawpy.plot.`**`plot_optical`**(*datafile: str = 'optical.h5', keys: List[str] = ['AbsorptionCoefficient', 'ExtinctionCoefficient', 'RefractiveIndex', 'Reflectance'], axes: List[str] = ['X', 'Y', 'Z', 'XY', 'YZ', 'ZX'], raw: bool = False, prefix: str = '', save: bool = True, verbose: bool = False*)

After the optical property calculation task is completed, read the data and draw a preview image

optical.h5/optical.json -> optical.png

**Parameters**

- **`datafile`** – Path to an h5 or json file, or a folder containing any of these files, default optical.h5
- **`keys`** – One of AbsorptionCoefficient, ExtinctionCoefficient, RefractiveIndex, Reflectance, default AbsorptionCoefficient
- **`axes`** – Index, default X, Y, Z, XY, YZ, ZX
- **`raw`** – Whether to save plot data to CSV
- **`prefix`** – Folder path to save images, if empty, saves in the current directory
- **`save`** – Whether to save the image, default is True

#### Examples

Plot and save the plot data to rawoptical.csv

```
>>> from dspawpy.plot import plot_optical
>>> plot_optical("dspawpy_proj/dspawpy_tests/inputs/2.12/scf.h5",
↪"AbsorptionCoefficient", ['X', 'Y'], prefix='dspawpy_proj/dspawpy_tests/outputs/
↪doctest')
>>> plot_optical("dspawpy_proj/dspawpy_tests/inputs/2.12/optical.json", [
↪"AbsorptionCoefficient"], ['X', 'Y'], prefix='dspawpy_proj/dspawpy_tests/outputs/
↪doctest', raw=True)
```

**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, it is likely that the program you are using (such as MobaXterm, etc.) is incompatible with the QT libraries. You can either switch to another program (such as VSCode or the systems built-in terminal command line) or add the following code to your Python script, starting from the second line:

```
import matplotlib
matplotlib.use('agg')
```

## 8.8 neb data processing

Lets start with a quick introduction using the H diffusion on Pt(100) surface example:

### 8.8.1 Generating intermediate configurations for input files

- See `8neb_interpolate_structures.py`:

```
# coding:utf-8
from dspawpy.diffusion.neb import NEB, write_neb_structures
from dspawpy.diffusion.nebtools import write_json_chain
from dspawpy.io.structure import read

# Read initial configuration
init_struct = read("dspawpy_proj/dspawpy_tests/inputs/2.15/00/structure00.as")[0]
# Read final state configuration
final_struct = read("dspawpy_proj/dspawpy_tests/inputs/2.15/04/structure04.as")[0]

neb = NEB(
    initial_structure=init_struct,  # Initial structure
    final_structure=final_struct,  # Final state configuration
    nimages=8,  # Total of 8 configurations, including initial and final states
)
structures = neb.linear_interpolate()  # Linear interpolation
# structures = neb.idpp_interpolate()   # IDPP interpolation

# Save as structure file to dest path
write_neb_structures(
    structures=structures,  # Insert interpolated structure chains
    coords_are_cartesian=True,  # Whether to save in Cartesian coordinates
    fmt="as",  # Save format, supported formats: 'json', 'as', 'hzw', 'pdb', 'xyz', 'dump'
    path="dspawpy_proj/dspawpy_tests/outputs/us/8neb_interpolate_structures",  # Save path
    prefix="structure",  # File name prefix
)

# Preview initial structure chain
write_json_chain(
    preview=True,  # whether to enable preview mode
    directory="dspawpy_proj/dspawpy_tests/outputs/us/8neb_interpolate_structures", # Directory for NEB calculations
    step=-1,  # Default to saving the structure chain of the last ion step (latest)
```

(continues on next page)

(continued from previous page)

```
    dst="dspawpy_proj/dspawpy_tests/outputs/us/8neb",  # Save path
)
# write_xyz_chain(preview=True, # Whether to run in preview mode
#                  directory="dspawpy_proj/dspawpy_tests/outputs/us/8neb_interpolate_structures", # NEB calculation directory
#                  step=-1, # Default to saving the structure chain of the last ionic step (latest)
#                  dst='dspawpy_proj/dspawpy_tests/outputs/us/8neb' # save path
# )
```

**Note**

1. Users can modify the number of interpolated points as needed. Setting it to 8 will generate a folder containing 8 structure files, with 6 intermediate configurations.
2. `neb.linear_interpolate` is a linear interpolation method. The `pbc` parameter, when set to `True`, will lock the search for the shortest diffusion path. It defaults to `False` to increase user control, because
3. For example, if the initial fractional coordinate of an atom is 0.2 and the final state is 0.8. When pbc = True, the diffusion path will be forced to be 0.2 -> -0.2. When pbc = False, the user can make the program perform interpolation along the diffusion path 0.2 -> 0.8; if the shortest path is desired, manually change 0.8 to -0.2, thereby ensuring the program completes the initial guess of interpolation according to the users intent.

## 8.8.2 Plotting the energy barrier diagram

### 8.8.2.1 neb.iniFin = true/false

When neb.iniFin = true/false, you can use the path from the NEB calculation for barrier analysis (ensure that the initial and final state calculation files are in the NEB calculation path):

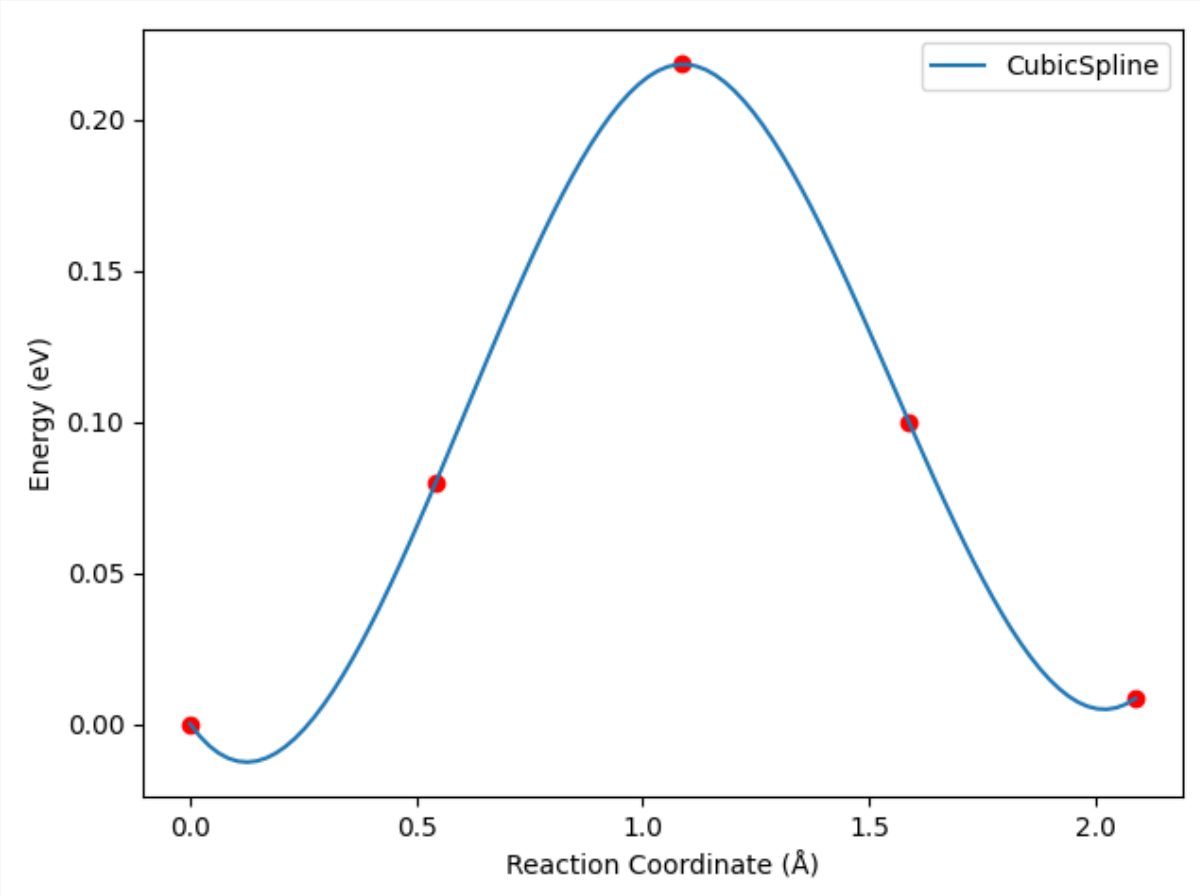
- Refer to `8neb_barrier_CubicSpline.py`:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import plot_barrier

directory_of_neb_task = (
    "dspawpy_proj/dspawpy_tests/inputs/2.15"  # <-- Please modify to the actual NEB path
)

# Plotting the energy barrier using CubicSpline interpolation
plot_barrier(
    directory=directory_of_neb_task,  # path of the neb task
    method="CubicSpline",  # Cubic spline interpolation
    figname="dspawpy_proj/dspawpy_tests/outputs/us/8neb_barrier_CubicSpline.png", # Output filename for the energy barrier plot
    show=False,  # Whether to display the energy barrier plot
)
```

**Note**

After running the above script, you can obtain a barrier curve similar to the following, with cubic spline

interpolation:




For this specific example, the curve will exhibit an undesirable dip after cubic spline interpolation, which is inherent to the characteristics of the cubic spline interpolation algorithm.

dspawpy internally calls scipys interpolation algorithms. Taking the cubic spline interpolation algorithm as an example in the script above, it is defined in the scipy documentation as:

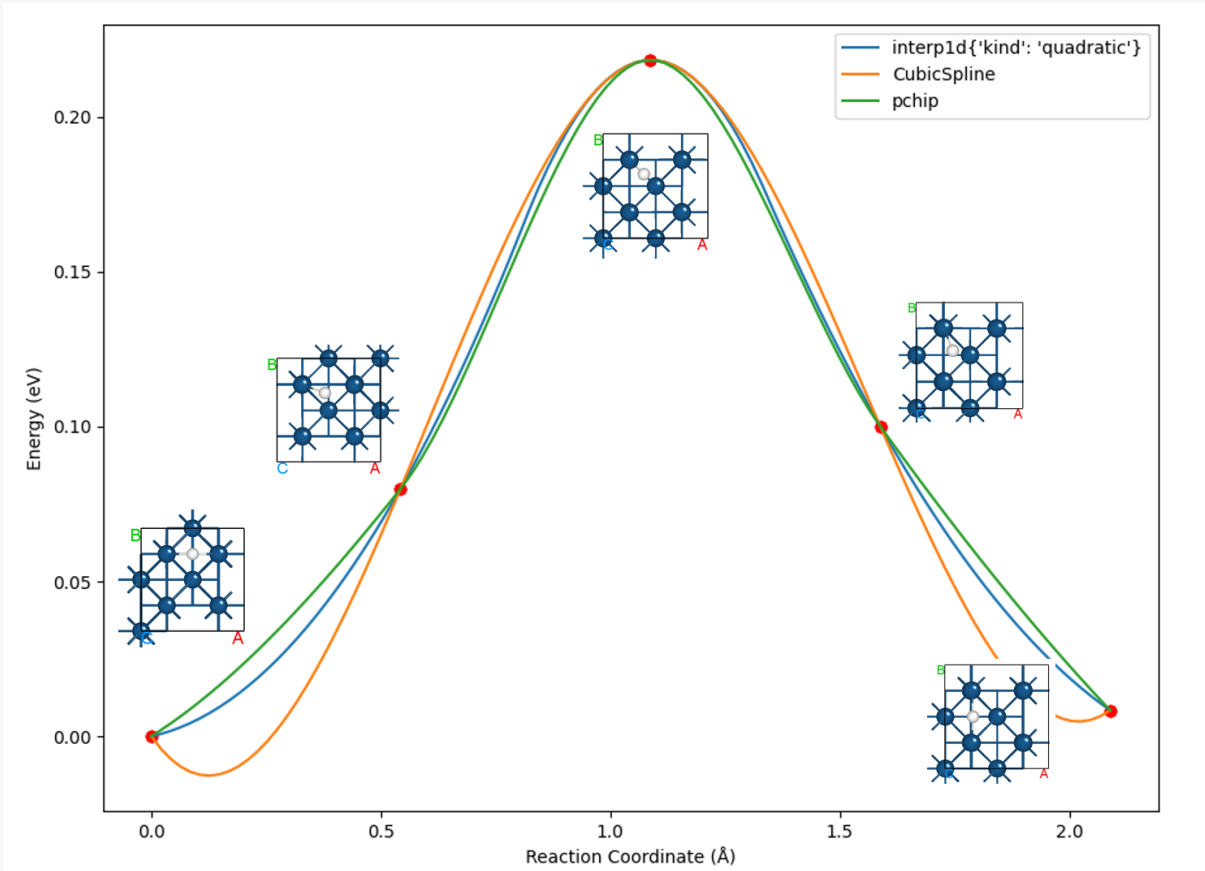
```
class scipy.interpolate.CubicSpline(x, y, axis=0, bc_type='not-a-knot',␣
↪extrapolate=None)
```

The keyword arguments include axis, bc_type, and extrapolate, whose specific meanings can be found in scipy.interpolate.CubicSpline. We can specify the corresponding keyword arguments (axis, bc_type, extrapolate) in the plot_barrier function and pass them to the scipy.interpolate.CubicSpline class for processing.

Here we use the script `8neb_barrier.py` to compare the curves plotted by interpolating with three algorithms:

```
# coding:utf-8
import os

import matplotlib.pyplot as plt

from dspawpy.diffusion.nebtools import plot_barrier

# Compare the differences in energy barrier curves drawn by different interpolation␣
```

```
↪methods, where show should be set to False
# 1. interp1d
plot_barrier(
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",  # path for NEB calculation
    ri=None,  # Reaction coordinate between the initial structure and the second␣
↪structure, required when the NEB task only calculated intermediate structures
    rf=None,  # Reaction coordinate between the last configuration and the second-to-
↪last configuration, when the NEB task only calculated intermediate configurations
    ei=None,  # Energy of the initial configuration, required when the NEB task only␣
↪calculated intermediate configurations
    ef=None,  # Energy of the final configuration, required when the NEB task only␣
↪calculated intermediate configurations
    method="interp1d",  # Interpolation method
    figname=None,  # Name of the output energy barrier plot file
    show=False,  # Whether to display the energy barrier plot
    kind="quadratic",  # Parameter of the interpolation method
)
# 2. CubicSpline
plot_barrier(
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",
    method="CubicSpline",
    figname=None,
    show=False,
)
# 3. pchip
plot_barrier(
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",
    method="pchip",
    figname=None,
    show=False,
)

filename = "dspawpy_proj/dspawpy_tests/outputs/us/8neb_barrier_comparison.png"  #␣
↪Filename for the energy barrier plot output
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
plt.savefig(filename, dpi=300)
# plt.show()
```




**Note**

1. Choosing the appropriate interpolation algorithm is crucial for optimizing the final curve presentation.
2. In most cases, selecting the pchip (piecewise cubic Hermite interpolating polynomial) monotonic cubic spline interpolation algorithm will achieve good results, and it is also the default interpolation algorithm called.

#### 8.8.2.2 neb.iniFin = true

When *neb.iniFin = true* is set, reading the *neb.h5/neb.json* files generated by the NEB calculation allows for a quick barrier analysis:

- See `8neb_barrier_CubicSpline.py`:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import plot_barrier

# Plot energy barrier using CubicSpline interpolation
plot_barrier(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.15/neb.h5",  # Path to neb.h5
    method="CubicSpline",  # Cubic spline interpolation
    figname="dspawpy_proj/dspawpy_tests/outputs/us/8neb_barrier_.png",  # Output␣
↪file name for the energy barrier plot
```

(continues on next page)

(continued from previous page)

```
    show=False,  # Whether to display the energy barrier plot
)
```

Processing the resulting barrier diagram is consistent with the previously read path.

**Note**

1. The energy stored in neb.h5 and neb.json files is TotalEnergy. If you need an accurate barrier value, it is recommended to process it by reading the NEB calculation path (taking TotalEnergy0).

**Warning**

If you are connecting to a remote server via SSH to execute the script above and encounter QT-related error messages, its possible that the program you are using (e.g., MobaXterm) is incompatible with the QT libraries. Either switch to a different program (such as VSCode or the systems built-in terminal command line), or add the following code starting on the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.8.3 Processing Data for Transition State Calculations

After NEB calculations, it is generally necessary to plot the energy barrier diagram and check the forces on each interpolated structure to ensure they are below a specified threshold. If the results are abnormal, the force and energy changes of each interpolated structure during the structure optimization process should also be checked to determine if they have truly converged. These operations require at least three cycles. To simplify the process, we provide an all-in-one summary function `summary`:

- Refer to `8neb_check_results.py`:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import summary

# Import the neb calculation directory, a complete folder after neb calculation␣
↪needs to be provided
summary(
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",
    show_converge=False,  # Whether to display the convergence plots of energy and␣
↪force
    outdir="dspawpy_proj/dspawpy_tests/outputs/us/8neb",  # Path to save␣
↪convergence plots of energy and forces
    figname="dspawpy_proj/dspawpy_tests/outputs/us/8neb/neb_barrier_summary.png",
↪# Path to save the energy barrier plot
)
# Additional keyword arguments can be set for plotting the barrier diagram, such as:
# summary(directory='dspawpy_proj/dspawpy_tests/inputs/2.15', method='CubicSpline') #␣
↪Change to CubicSpline for spline interpolation
```

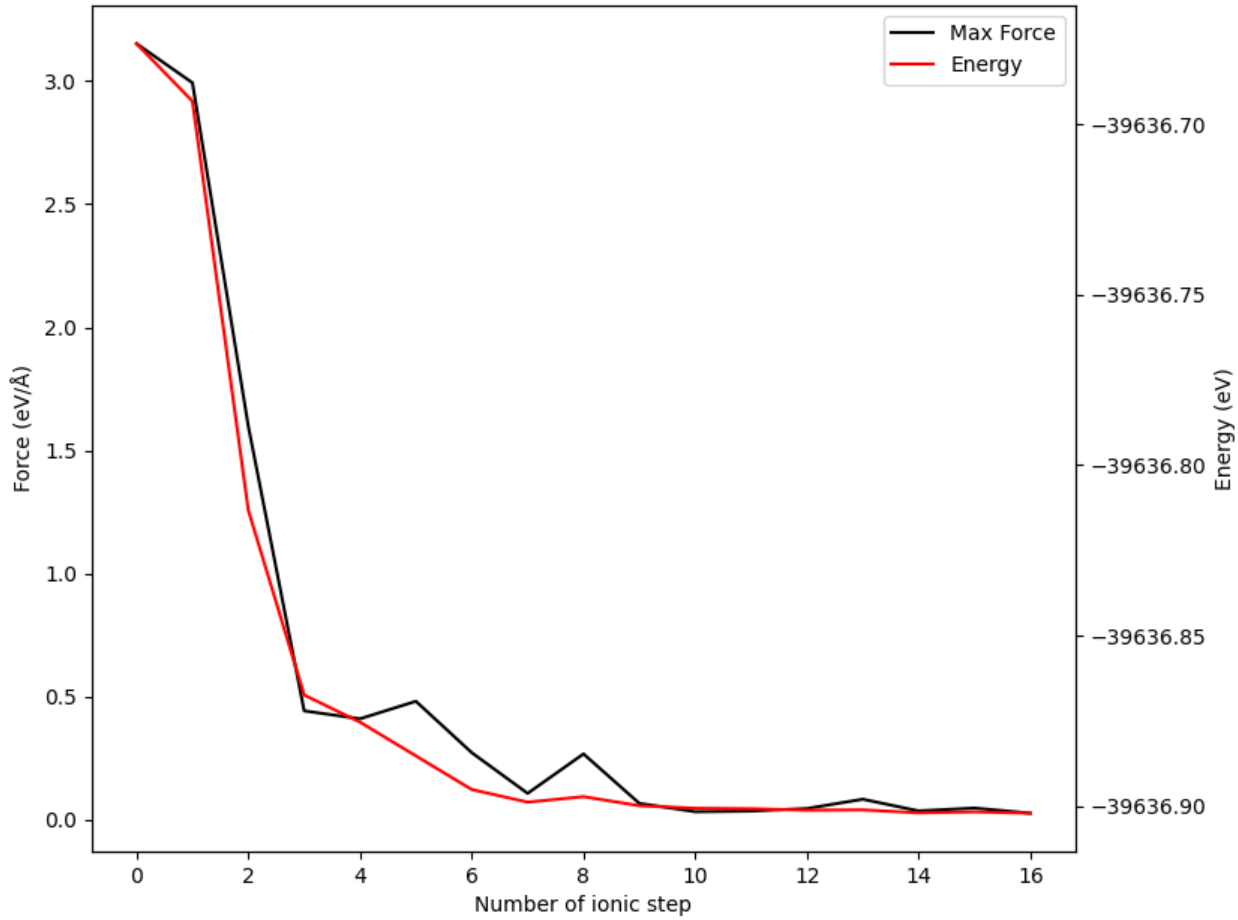
**Note**

1. This script will print the energies and forces of each structure in a table, plot the energy barrier, and also plot the convergence of energy and forces for intermediate structures.
2. If `neb.iniFin = false`, the user must copy the results file of the self-consistent calculation, either *scf.h5* or *system.json*, to the corresponding initial and final state subfolders. Otherwise, the program cannot read the energy and force information of the initial and final states and will exit with an error.
3. By default, the energy barrier plot is stored in the parent directory of the NEB calculation, and the energy and force convergence plots for each intermediate structure are stored in the respective subfolders.

Executing the code will generate a table similar to the following, displaying the energy and force information for each NEB configuration:

| Image | Force (eV/Å) | Reaction coordinate (Å) | Energy (eV) | Delta energy (eV) |
|---|---|---|---|---|
| 00 | 0.1803 | 0.0000 | -39637.0984 | 0.0000 |
| 01 | 0.0263 | 0.5428 | -39637.0186 | 0.0798 |
| 02 | 0.0248 | 1.0868 | -39636.8801 | 0.2183 |
| 03 | 0.2344 | 1.5884 | -39636.9984 | 0.1000 |
| 04 | 0.0141 | 2.0892 | -39637.0900 | 0.0084 |

In addition to the energy barrier diagram, you can also obtain the energy and force convergence curves for each intermediate configuration (taking configuration 02 as an example).




**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, it might be due to incompatibility between the program you are using (such as MobaXterm) and the QT library. Either switch to another program (such as VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.8.4 Observing the NEB Chain

Here, the NEB chain refers to the geometric relationship between the interpolated structures (structure00.as, structure01.as, ), rather than the changes of a single structure during the optimization process.

- NEB calculations are computationally expensive, and observing the NEB chain helps to judge the convergence speed of the NEB calculation. Furthermore, after generating intermediate structures via interpolation, previewing the NEB chain is often necessary. These needs can be met using the `8neb_visualize.py` script:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import write_json_chain, write_xyz_chain

```

(continues on next page)

(continued from previous page)

```
# Convert the configuration chain under the NEB calculation path to a JSON format file
write_json_chain(
    preview=False,  # If the NEB calculation is already completed, preview mode is not required
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",  # NEB calculation directory
    step=-1,  # Default to saving the configuration chain of the last ion step (latest)
    dst="dspawpy_proj/dspawpy_tests/outputs/us/8neb",  # Save path
    ignorels=False,  # Set to True to ignore latestStructureXX.as files
)

# Convert the configuration chain in the NEB calculation path to xyz format files
write_xyz_chain(
    preview=False,  # If the NEB calculation is already completed, preview mode is not required
    directory="dspawpy_proj/dspawpy_tests/inputs/2.15",  # NEB calculation directory
    step=-1,  # Default to saving the configuration chain of the last ionic step (latest)
    dst="dspawpy_proj/dspawpy_tests/outputs/us/8neb",  # Save path
    ignorels=False,  # Set to True to ignore latestStructureXX.as files
)
```

**Note**

1. After this script generates the neb_movie*.json files, you can view them by opening the json file via `Device Studio` –> `Simulator` –> `DS-PAW` –> `Analysis Plot`.
2. The *directory* parameter specifies the main path of the NEB calculation; the complete folder after the NEB calculation is finished must be provided.
3. This script supports processing ongoing (i.e., incomplete) NEB calculation files, allowing users to monitor the trajectory in real time.
4. The xyz file can be opened and viewed using OVITO software: Open the visualization interface via `Device Studio` –> `Simulator` –> `OVITO`, and then drag and drop the xyz file.
5. Structure information reading priority: latestStructureXX.as > h5 > json; When ignorels is set to True, it first attempts to read data from h5, and if it fails, it reads from json.

### 8.8.5 Calculate the inter-configuration distance

- Refer to this script: `8calc_dist.py`:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import get_distance
from dspawpy.io.structure import read

# Please modify the paths of structure01.as and structure02.as structure files according to the actual situation
# First read the fractional coordinates, element list, and cell information of the two configurations
s1 = read("dspawpy_proj/dspawpy_tests/inputs/2.15/01/structure01.as")[0]
s2 = read("dspawpy_proj/dspawpy_tests/inputs/2.15/02/structure02.as")[0]
# Calculate the distance between the two configurations, note that this function only accepts fractional coordinates
```

(continues on next page)

(continued from previous page)

```
dist = get_distance(
    spo1=s1.frac_coords,
    spo2=s2.frac_coords,
    lat1=s1.lattice.matrix,
    lat2=s2.lattice.matrix,
)
print("The distance between the two configurations is:", dist, "Angstrom")
```

### 8.8.6 Continued calculation with neb

- To restart a NEB calculation, refer to `8neb_restart.py`:

```
# coding:utf-8
import os
from shutil import copytree, rmtree

from dspawpy.diffusion.nebtools import restart

if os.path.isdir("dspawpy_proj/dspawpy_tests/outputs/us/neb4bk"):
    rmtree("dspawpy_proj/dspawpy_tests/outputs/us/neb4bk")

copytree(
    "dspawpy_proj/dspawpy_tests/inputs/2.15",
    "dspawpy_proj/dspawpy_tests/outputs/us/neb4bk",
)
restart(
    directory="dspawpy_proj/dspawpy_tests/outputs/us/neb4bk",  # NEB task path
    output="dspawpy_proj/dspawpy_tests/outputs/us/8neb_restart",  # Backup␣
↪destination
)
```

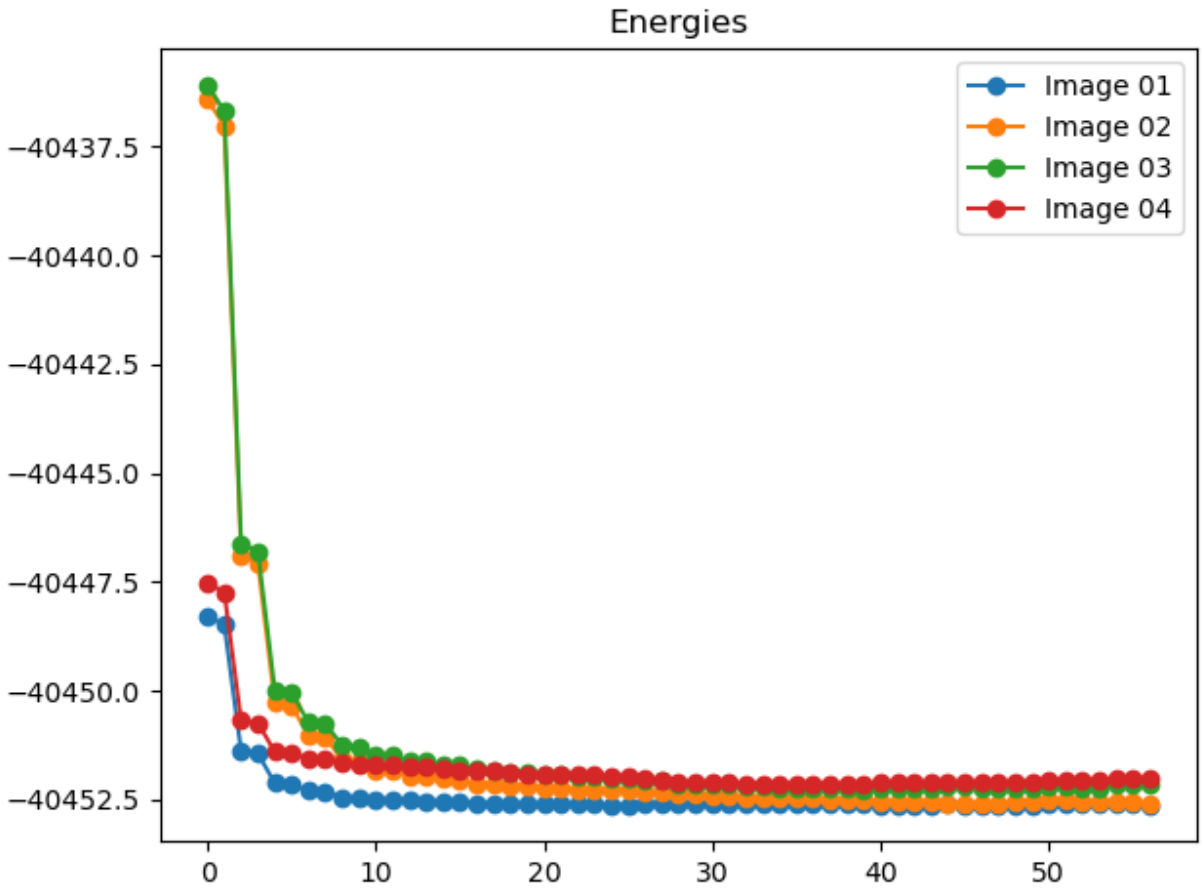
See *Continued calculation with neb* for details.

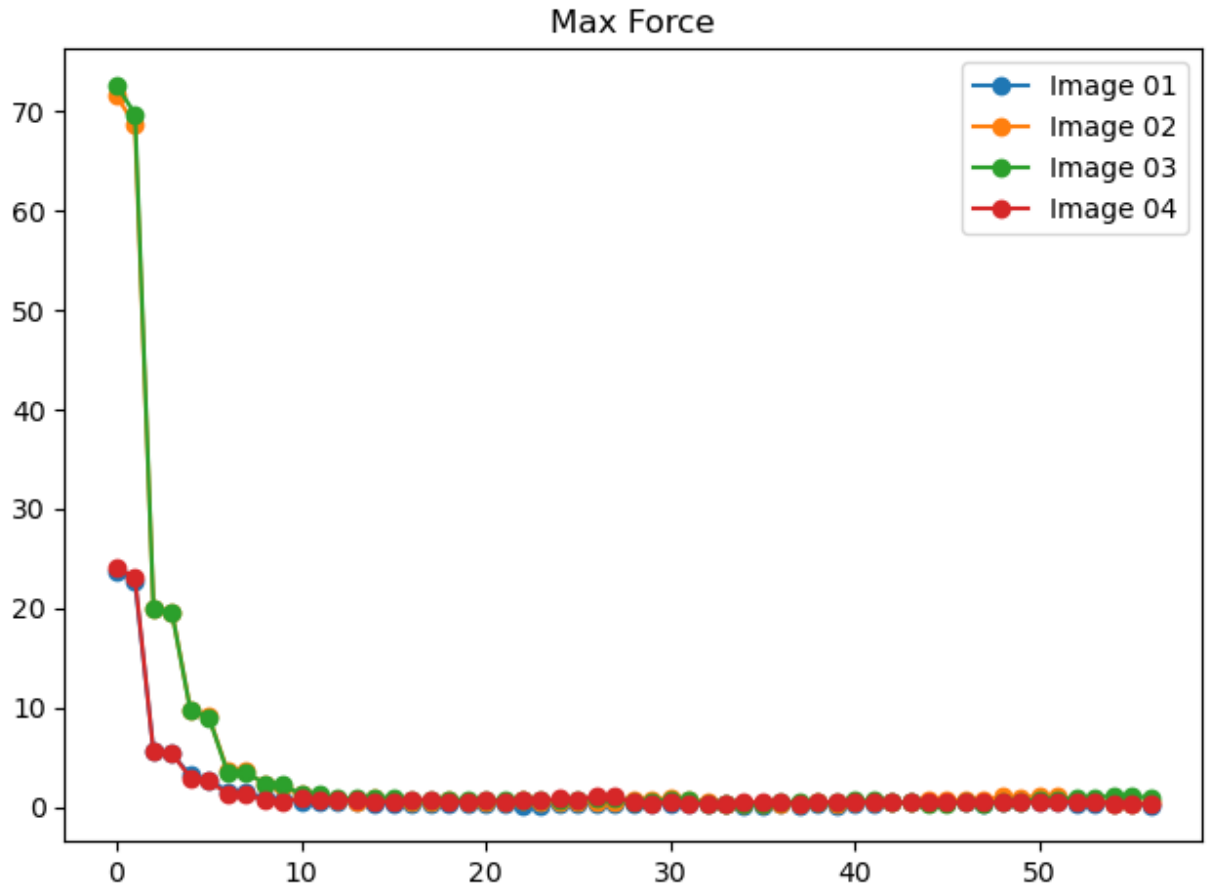
### 8.8.7 Energy and maximum atomic force variation trend during NEB calculation

- To view plots showing the energy and maximum atomic force trends during the NEB calculation, refer to `8neb_energy_force_curves.py`:

```
# coding:utf-8
from dspawpy.diffusion.nebtools import monitor_force_energy

# Specify the path to the NEB calculation folder; after running, Energies.png and␣
↪MaxForce.png images will be generated in the specified directory
unfinished_neb_folder = "dspawpy_proj/dspawpy_tests/inputs/supplement/neb_unfinished
↪"
monitor_force_energy(
    directory=unfinished_neb_folder,
    outdir="imgs",  # Output image path
)
```

Generates energy and force change trend charts:


Energies
Image 01
Image 02
Image 03
Image 04
−40437.5
−40440.0
−40442.5
−40445.0
−40447.5
−40450.0
−40452.5
0
10
20
30
40
50




**API: write_neb_structures(), plot_barrier(), summary(), get_distance(), write_movie_json(), write_xyz(), restart()**

- The *write_neb_structures* function is responsible for generating intermediate configurations:

  `dspawpy.diffusion.neb.write_neb_structures`(*structures: list*, *coords_are_cartesian: bool = True*, *fmt: str = 'as'*, *path: str = '.'*, *prefix='structure'*)

  Interpolate and generate intermediate configuration files

  **Parameters**

  - **structures** – Structure list
  - **coords_are_cartesian** – Is the coordinate Cartesian
  - **fmt** – Structure file type, default to as
  - **path** – Save path
  - **prefix** – Filename prefix, default to structure, which will generate files like structure00.as, structure01.as,

  **Returns**
  Saves the configuration file

  **Return type**
  file

### Examples

First, read the .as file to create a structure object

```
>>> from dspawpy.io.structure import read
>>> init_struct = read("dspawpy_proj/dspawpy_tests/inputs/2.15/00/structure00.as
↪")[0]
>>> final_struct = read("dspawpy_proj/dspawpy_tests/inputs/2.15/04/structure04.
↪as")[0]
```

Then, interpolate and generate intermediate structure files

```
>>> from dspawpy.diffusion.neb import NEB,write_neb_structures
>>> neb = NEB(init_struct,final_struct,8)
>>> structures = neb.linear_interpolate()   # Linear interpolation
```

Interpolated structures can be saved to the neb folder.

```
>>> write_neb_structures(structures, path="dspawpy_proj/dspawpy_tests/outputs/
↪doctest/11neb_interpolate_structures")
==> ...structure00.as...
==> ...structure01.as...
==> ...structure02.as...
==> ...structure03.as...
==> ...structure04.as...
==> ...structure05.as...
==> ...structure06.as...
==> ...structure07.as...
```

- The *plot_barrier* function is responsible for plotting the energy barrier diagram:

`dspawpy.diffusion.nebtools.`**`plot_barrier`**(*datafile: str = 'neb.h5', directory: str | None = None, ri: float | None = None, rf: float | None = None, ei: float | None = None, ef: float | None = None, method: str = 'PchipInterpolator', figname: str | None = 'neb_barrier.png', show: bool = True, raw: bool = False, verbose: bool = False, **kwargs*)

Call the scipy.interpolate interpolation algorithm to fit the NEB barrier and plot

**Parameters**

- **datafile** – Path to neb.h5 or neb.json file
- **directory** – NEB calculation path
- **ri** – Initial reaction coordinate
- **rf** – Final state reaction coordinate
- **ei** – Initial state self-consistent energy
- **ef** – Final state self-consistent energy
- **method** (*str, optional*) – Interpolation algorithm, default PchipInterpolator
- **figname** (*str, optional*) – Barrier image name, default neb_barrier.png
- **show** (*bool, optional*) – Whether to display the interactive interface, default True
- **raw** (*bool, optional*) – Whether to return plotting data to CSV

**Raises**

- **ImportError** – The specified interpolation algorithm does not exist in scipy.interpolate
- **ValueError** – The parameters passed to the interpolation algorithm do not meet the requirements of the algorithm

### Examples

```
>>> from dspawpy.diffusion.nebtools import plot_barrier
>>> import matplotlib.pyplot as plt
```

Comparing different interpolation algorithms

```
>>> plot_barrier(directory='dspawpy_proj/dspawpy_tests/inputs/2.15', method=
↪'interp1d', kind=2, figname=None, show=False)
>>> plot_barrier(directory='dspawpy_proj/dspawpy_tests/inputs/2.15', method=
↪'interp1d', kind=3, figname=None, show=False)
>>> plot_barrier(directory='dspawpy_proj/dspawpy_tests/inputs/2.15', method=
↪'CubicSpline', figname=None, show=False)
>>> plot_barrier(directory='dspawpy_proj/dspawpy_tests/inputs/2.15', method=
↪'pchip', figname='dspawpy_proj/dspawpy_tests/outputs/doctest/barrier_
↪comparison.png', show=False)
==> ...barrier_comparison.png...
```

Attempt to read neb.h5 file or neb.json file

```
>>> plot_barrier(datafile='dspawpy_proj/dspawpy_tests/inputs/2.15/neb.h5',␣
↪method='pchip', figname='dspawpy_proj/dspawpy_tests/outputs/doctest/barrier_
↪h5.png', show=False)
==> ...barrier_h5.png
>>> plot_barrier(datafile='dspawpy_proj/dspawpy_tests/inputs/2.15/neb.json',␣
↪method='pchip', figname='dspawpy_proj/dspawpy_tests/outputs/doctest/barrier_
↪json.png', show=False)
==> ...barrier_json.png...
```

- The `summary` function is responsible for summarizing the NEB calculation tasks documentation:

`dspawpy.diffusion.nebtools.`**`summary`**(*directory: str = '.', raw=False, show_converge=False, outdir: str | None = None, **kwargs*)

Summary of NEB task completion, execute the following steps in order:

- 1. Print the forces, reaction coordinates, energy, and energy differences from the initial configuration for each structure
- 2. Plot the energy barrier diagram
- 3. Plot and save the convergence processes of energy and forces during the structure optimization

**Parameters**

- **directory** – NEB path, default to the current path
- **raw** – Whether to save the plot data to a CSV file
- **show_converge** – Whether to display energy and force convergence plots of the structural optimization process, default is not displayed
- **outdir** – Path to save the convergence process figure, default to directory

– ****kwargs** (*dict*) – Parameters passed to plot_barrier

### Examples

```
>>> from dspawpy.diffusion.nebtools import summary
>>> directory = 'dspawpy_proj/dspawpy_tests/inputs/2.15' # Path for NEB␣
↪calculation, default to current path
>>> summary(directory, show=False, figname='dspawpy_proj/dspawpy_tests/outputs/
↪doctest/neb_barrier.png')
shape: (5, 5)

 FolderName  Force(eV/Å)  RC(Å)     Energy(eV)     E-E0(eV)

 00          0.180272     0.0       -39637.097656  0.0
 01          0.014094     0.542789  -39637.019531  0.079814
 02          0.026337     1.0868    -39636.878906  0.218265
 03          0.024798     1.588367  -39637.0       0.100043
 04          0.234429     2.089212  -39637.089844  0.008414

==> ...neb_barrier.png...
==> ...converge.png...
==> ...converge.png...
==> ...converge.png...
```

```
>>> summary(directory, show=False, figname='dspawpy_proj/dspawpy_tests/outputs/
↪doctest/neb_barrier.png', outdir="dspawpy_proj/dspawpy_tests/outputs/doctest/
↪neb_summary")
shape: (5, 5)

 FolderName  Force(eV/Å)  RC(Å)     Energy(eV)     E-E0(eV)

 00          0.180272     0.0       -39637.097656  0.0
 01          0.014094     0.542789  -39637.019531  0.079814
 02          0.026337     1.0868    -39636.878906  0.218265
 03          0.024798     1.588367  -39637.0       0.100043
 04          0.234429     2.089212  -39637.089844  0.008414

==> ...neb_barrier.png...
==> ...converge.png...
==> ...converge.png...
==> ...converge.png...
```

If inifin=False, the user must place a converged scf.h5 or system.json in the initial and final state subfolders.

- The `get_distance` function calculates the distance between two configurations:

  `dspawpy.diffusion.nebtools.`**`get_distance`**(*spo1*, *spo2*, *lat1*, *lat2*)

  Calculate the distance between two structures based on their fractional coordinates and cell parameters

  **Parameters**

  – **spo1** (*np.ndarray*) – Scores coordinate list 1

  – **spo2** (*np.ndarray*) – Fractional coordinate list 2

  – **lat1** (*np.ndarray*) – Cell 1

– **lat2** (*np.ndarray*) – Cell 2

**Returns**
Distance

**Return type**
float

### Examples

First, read the structure information

```
>>> from dspawpy.io.structure import read
>>> s1 = read('dspawpy_proj/dspawpy_tests/inputs/2.15/01/structure01.as')[0]
>>> s2 = read('dspawpy_proj/dspawpy_tests/inputs/2.15/02/structure02.as')[0]
```

Calculate the distance between two configurations

```
>>> from dspawpy.diffusion.nebtools import get_distance
>>> dist = get_distance(s1.frac_coords, s2.frac_coords, s1.lattice.matrix, s2.
↪lattice.matrix)
>>> print('The distance between the two configurations is:', dist, 'Angstrom')
The distance between the two configurations is: 0.476972808803491 Angstrom
```

- The functions `write_movie_json` and `write_xyz` can write intermediate configurations to JSON or XYZ files:
- The `restart` function is responsible for restarting the NEB calculation:

`dspawpy.diffusion.nebtools.`**`restart`**(*directory: str = '.', output: str = 'bakfile'*)

Archive and compress old NEB tasks, and prepare for continuation at the original path

**Parameters**

– **directory** – Old NEB task path, default current path

– **output** – Backup folder path, default is to create a bakfile folder in the current path for backup; Alternatively, you can specify any path, but it cannot be the same as the current path

### Examples

```
>>> from dspawpy.diffusion.nebtools import restart
>>> from shutil import copytree
>>> copytree('dspawpy_proj/dspawpy_tests/inputs/2.15', 'dspawpy_proj/dspawpy_
↪tests/outputs/doctest/neb4bk2', dirs_exist_ok=True)
'dspawpy_proj/dspawpy_tests/outputs/doctest/neb4bk2'
>>> restart(directory='dspawpy_proj/dspawpy_tests/outputs/doctest/neb4bk2',␣
↪output='dspawpy_proj/dspawpy_tests/outputs/doctest/neb_backup')
==> ...neb_backup...
```

The preparation for the continuation calculation may take a long time to complete, please be patient

- The `monitor_force_energy` function is responsible for plotting the energy and force changes during the NEB calculation:

`dspawpy.diffusion.nebtools.`**`monitor_force_energy`**(*directory: str, outdir: str = '.', relative: bool = False*)

Read forces and energies during NEB calculations from xx/DS-PAW.log and plot curves

No JSON files are output during the calculation, and only force information is present in nebXX.h5 files, so DS-PAW.log must be read.

Energy matching mode, should hit -40521.972259

#### 8.8.7.1 LOOP 1:

# iter | Etot(eV) dE(eV) time # 1 | -35958.655378 -3.595866e+04 47.784 s # 2 | -40069.322436 -4.110667e+03 15.146 s # 3 | -40490.281166 -4.209587e+02 15.114 s # 4 | -40521.972259 -3.169109e+01 17.936 s

### Examples

```
>>> from dspawpy.diffusion.nebtools import monitor_force_energy
>>> monitor_force_energy(
...     directory="dspawpy_proj/dspawpy_tests/inputs/supplement/neb_unfinished",
...     outdir="imgs"
... )
Max Force shape: (57, 4)

 Folder 01  Folder 02  Folder 03  Folder 04

 23.775228  71.547767  72.641234  24.147289
 22.683711  68.595607  69.704747  23.0549
 5.624252   20.071221  20.049429  5.567894
 5.354774   19.631643  19.599093  5.425462
 3.188546   9.840143   9.748006   2.943709

 0.293867   0.812679   0.920251   0.573649
 0.27249    0.7475     0.921836   0.540239
 0.299767   0.360673   1.174016   0.416171
 0.249903   0.288985   1.169237   0.366117
 0.204396   0.518356   0.913792   0.300884

Energies shape: (57, 4)

 Folder 01      Folder 02      Folder 03      Folder 04

 -40448.281556  -40436.419243  -40436.084611  -40447.527434
 -40448.491374  -40437.026948  -40436.685178  -40447.73947
 -40451.391617  -40446.884408  -40446.613158  -40450.686918
 -40451.448662  -40447.079933  -40446.803281  -40450.743777
 -40452.126865  -40450.274376  -40449.978142  -40451.405157

 -40452.620987  -40452.538682  -40452.230568  -40452.056262
 -40452.621777  -40452.544298  -40452.231776  -40452.055815
 -40452.620701  -40452.565649  -40452.164604  -40452.035357
 -40452.621371  -40452.569113  -40452.164784  -40452.037426
 -40452.622418  -40452.577864  -40452.141919  -40452.037885

==> ...MaxForce.png...
==> ...Energies.png...
```

## 8.9 Phonon Data Processing

Using the example of a phonon band structure and density of states calculation for MgO, using *phonon.h5*:

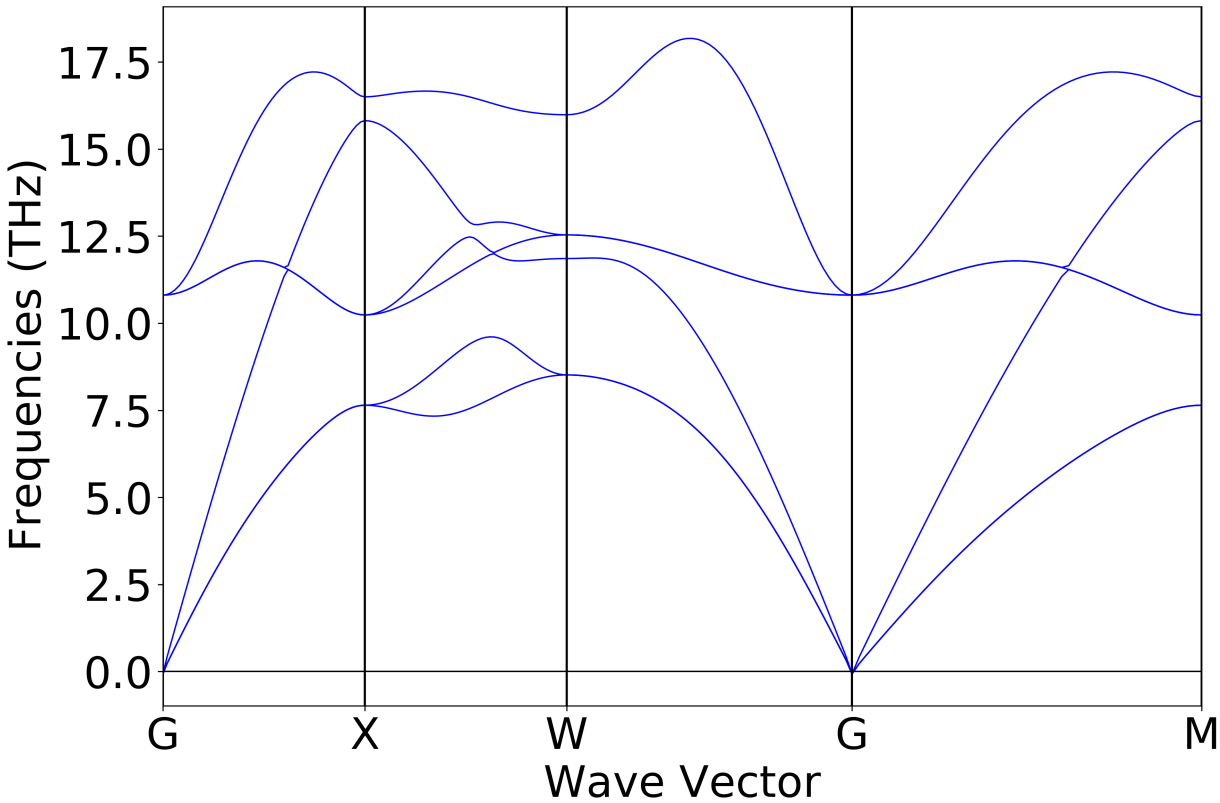
If phonopy is not installed, running the following script will result in the message `no module named 'phonopy'`, but this does not affect the programs normal operation.

### 8.9.1 Phonon band data processing

- Refer to `9phonon_bandplot.py`:

```python
# coding:utf-8
import os

from pymatgen.phonon.plotter import PhononBSPlotter

from dspawpy.io.read import get_phonon_band_data

band_data = get_phonon_band_data(
    "dspawpy_proj/dspawpy_tests/inputs/2.16.1/phonon.h5",
)  # Read phonon band structure
bsp = PhononBSPlotter(band_data)
axes_or_plt = bsp.get_plot(ylim=None, units="thz")  # Y-axis range # Units
import matplotlib.pyplot as plt  # noqa: E402

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif isinstance(axes_or_plt, tuple):
    fig = axes_or_plt[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

filename = "dspawpy_proj/dspawpy_tests/outputs/us/9phonon_bandplot.png"  # File name for the output phonon band plot
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
fig.savefig(filename, dpi=300)
```

Executing the code yields a phonon band structure curve similar to the following:




**Warning**

If you encounter QT-related error messages when executing the above script via SSH connection to a remote server, its likely due to incompatibility between the program used (e.g., MobaXterm) and the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code to your Python script starting from the second line:

```
import matplotlib
matplotlib.use('agg')
```

### 8.9.2 Phonon Density of States Data Processing

- Refer to `9phonon_dosplot.py`:

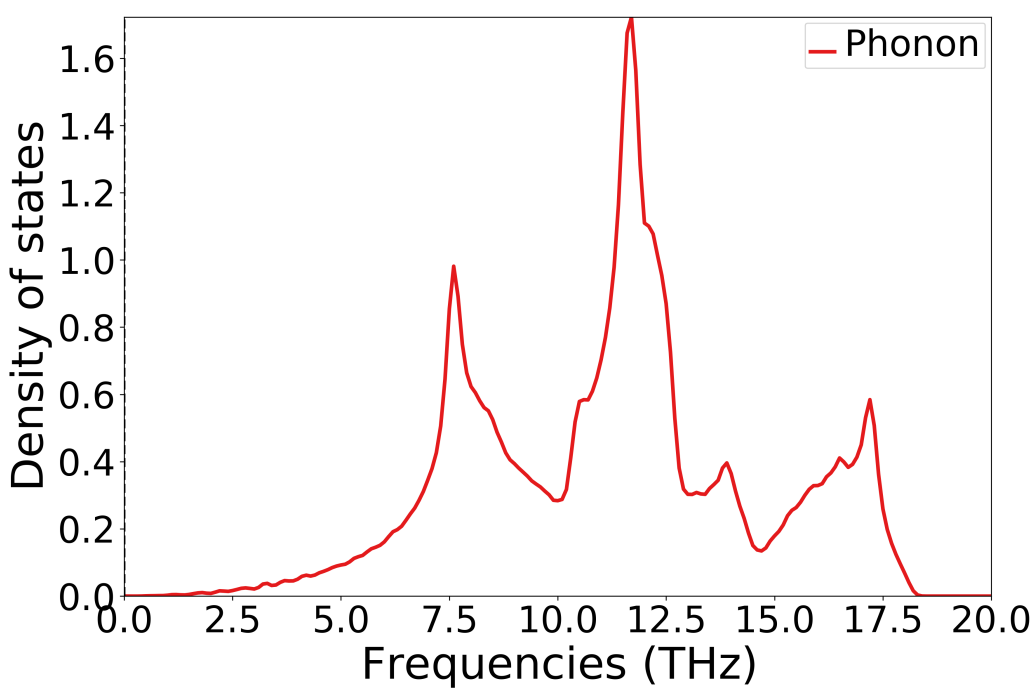
```
# coding:utf-8
import os

from pymatgen.phonon.plotter import PhononDosPlotter

from dspawpy.io.read import get_phonon_dos_data

dos = get_phonon_dos_data("dspawpy_proj/dspawpy_tests/inputs/2.16.1/phonon.h5")
dp = PhononDosPlotter(
    stack=False,  # True indicates drawing an area plot
```

(continues on next page)

(continued from previous page)

```
    sigma=None,  # Gaussian blur parameter
)
dp.add_dos(
    label="Phonon", dos=dos
)  # Legend  # The phonon density of states to be plotted
axes_or_plt = dp.get_plot(
    xlim=[0, 20],  # x-axis range
    ylim=None,  # y-axis range
    units="THz",  # Unit
)
import matplotlib.pyplot as plt  # noqa: E402

if isinstance(axes_or_plt, plt.Axes):
    fig = axes_or_plt.get_figure()  # version newer than v2023.8.10
elif isinstance(axes_or_plt, tuple):
    fig = axes_or_plt[0].get_figure()
else:
    fig = axes_or_plt.gcf()  # older version pymatgen

filename = " dspawpy_proj/dspawpy_tests/outputs/us/9phonon_dosplot.png"  # Energy␣
↪barrier plot output filename
os.makedirs(os.path.dirname(os.path.abspath(filename)), exist_ok=True)
fig.savefig(filename, dpi=300)
```

Executing the code yields a phonon density of states curve similar to the following:




**Warning**

If you execute the script above by SSH connection to a remote server and encounter QT-related error messages, its possible that the program you are using (such as MobaXterm) is incompatible with the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

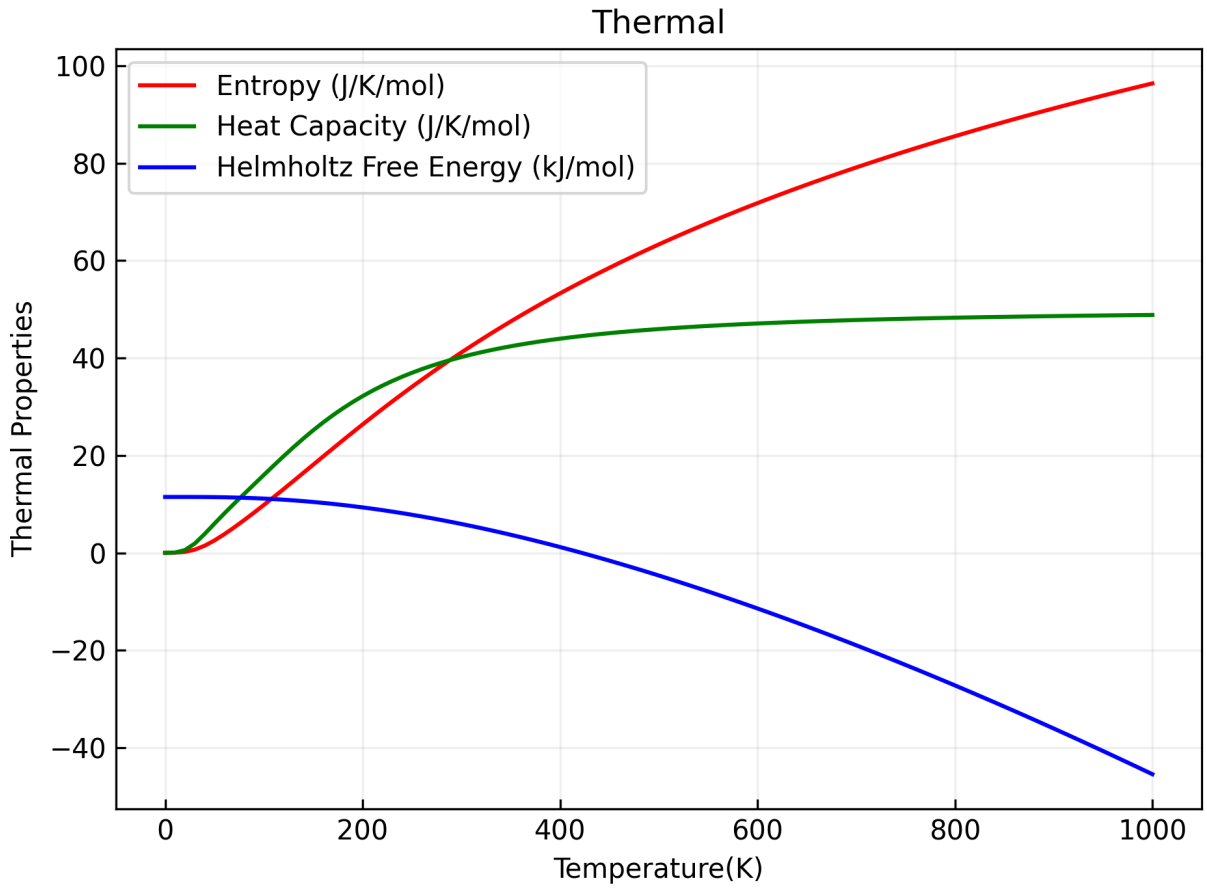
```
import matplotlib
matplotlib.use('agg')
```

### 8.9.3 Phonon Thermodynamic Data Processing

Refer to `9phonon_thermal.py`:

```
# coding:utf-8
from dspawpy.plot import plot_phonon_thermal

plot_phonon_thermal(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.26/phonon.h5",  # phonon.h5 data file␣
↪path
    figname="dspawpy_proj/dspawpy_tests/outputs/us/9phonon.png",  # Output phonon␣
↪thermodynamics figure filename
    show=False,  # Whether to display the image
)
```

Executing the code yields phonon thermodynamic curves similar to the following:




### API: get_phonon_band_data(), get_phonon_dos_data(), plot_phonon_thermal()

- The `get_phonon_band_data` function is responsible for reading phonon band data:

  `dspawpy.io.read.`**`get_phonon_band_data`**(*phonon_band_dir: str*, *verbose: bool = False*)

  Reads phonon band data from an h5 or json file and constructs a PhononBandStructureSymmLine object

  **Parameters**

  **phonon_band_dir** – Path to the band structure file, phonon.h5 / phonon.json, or a folder containing these files

  **Return type**

  PhononBandStructureSymmLine

  #### Examples

```
>>> from dspawpy.io.read import get_phonon_band_data
>>> band_data = get_phonon_band_data("dspawpy_proj/dspawpy_tests/inputs/2.16/
↪phonon.h5") # Read phonon band data
>>> band_data = get_phonon_band_data("dspawpy_proj/dspawpy_tests/inputs/2.16/
↪phonon.json") # Read phonon band data
```

- The `get_phonon_dos_data` function is responsible for reading the phonon density of states:

`dspawpy.io.read.`**`get_phonon_dos_data`**(*phonon_dos_dir: str*, *verbose: bool = False*)

Reads phonon density of states data from an h5 or json file, constructs a PhononDos object

**Parameters**

**phonon_dos_dir** – Path to the phonon DOS file, phonon_dos.h5 / phonon_dos.json, or a folder containing these files

**Return type**

PhononDos

### Examples

```
>>> from dspawpy.io.read import get_phonon_dos_data
>>> phdos = get_phonon_dos_data(phonon_dos_dir='dspawpy_proj/dspawpy_tests/
↪inputs/2.16.1/phonon.json')
>>> phdos = get_phonon_dos_data(phonon_dos_dir='dspawpy_proj/dspawpy_tests/
↪inputs/2.16.1/phonon.h5')
>>> phdos.frequencies
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,
        5.5,  5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,
        6.6,  6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7.4,  7.5,  7.6,
        7.7,  7.8,  7.9,  8. ,  8.1,  8.2,  8.3,  8.4,  8.5,  8.6,  8.7,
        8.8,  8.9,  9. ,  9.1,  9.2,  9.3,  9.4,  9.5,  9.6,  9.7,  9.8,
        9.9, 10. , 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 10.9,
       11. , 11.1, 11.2, 11.3, 11.4, 11.5, 11.6, 11.7, 11.8, 11.9, 12. ,
       12.1, 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, 12.9, 13. , 13.1,
       13.2, 13.3, 13.4, 13.5, 13.6, 13.7, 13.8, 13.9, 14. , 14.1, 14.2,
       14.3, 14.4, 14.5, 14.6, 14.7, 14.8, 14.9, 15. , 15.1, 15.2, 15.3,
       15.4, 15.5, 15.6, 15.7, 15.8, 15.9, 16. , 16.1, 16.2, 16.3, 16.4,
       16.5, 16.6, 16.7, 16.8, 16.9, 17. , 17.1, 17.2, 17.3, 17.4, 17.5,
       17.6, 17.7, 17.8, 17.9, 18. , 18.1, 18.2, 18.3, 18.4, 18.5, 18.6,
       18.7, 18.8, 18.9, 19. , 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7,
       19.8, 19.9, 20. ])
```

- The `plot_phonon_thermal` function is responsible for plotting phonon thermodynamic properties:

`dspawpy.plot.`**`plot_phonon_thermal`**(*datafile: str = 'phonon.h5'*, *figname: str = 'phonon.png'*, *show: bool = True*, *raw: bool = False*, *verbose: bool = False*)

Task completed for phonon thermodynamic calculations, plot curves of relevant physical quantities versus temperature

phonon.h5/phonon.json -> phonon.png

**Parameters**

- **datafile** – Path to an h5 or json file or a folder containing any of these files, default phonon.h5
- **figname** – Filename to save the image
- **show** – Whether to pop up an interactive interface
- **raw** – Whether to save the plotting data to rawphonon.csv file

**Returns**
Image path, default phonon.png

**Return type**
figname

### Examples

```
>>> from dspawpy.plot import plot_phonon_thermal
>>> plot_phonon_thermal('dspawpy_proj/dspawpy_tests/inputs/2.26/phonon.h5',␣
↪figname='dspawpy_proj/dspawpy_tests/outputs/doctest/phonon_thermal_h5.png',␣
↪show=False)
>>> plot_phonon_thermal('dspawpy_proj/dspawpy_tests/inputs/2.26/phonon.json',␣
↪figname='dspawpy_proj/dspawpy_tests/outputs/doctest/phonon_thermal_json.png',␣
↪show=False, raw=True)
```

**Warning**

If you execute the above script by connecting to a remote server via SSH and encounter QT-related error messages, its likely that the program you are using (e.g., MobaXterm) is incompatible with the QT libraries. You can either switch programs (e.g., VSCode or the systems built-in terminal) or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

## 8.10 aimd molecular dynamics simulation data processing

For a quick start, take the molecular dynamics simulation data of the $H_2O$ molecular system, for example, the *aimd.h5* file:

### 8.10.1 Convert the trajectory file format to .xyz or .dump.

Read data from the HDF5 file output by AIMD and generate trajectory files.

The generated .xyz or .dump files can be dragged and dropped into OVITO for visualization. You can open the OVITO visualization interface through Device Studio –> Simulator –> OVITO, and then drag and drop the .xyz or .dump files into OVITO.

See `10write_aimd_traj.py`:

```
# coding:utf-8
from dspawpy.io.structure import convert

convert(
    infile="dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5",  # Structure to be␣
↪converted, if in the current path, only the filename can be written
    si=None,  # Filter the configuration number, if not specified, read all by default
    ele=None,  # Filter element symbol, default reads atomic information for all elements
    ai=None,  # Filter atomic indices (starting from 1), default to read all atomic␣
↪information
    outfile="dspawpy_proj/dspawpy_tests/outputs/us/10aimdTraj.xyz",  # Can also generate␣
```

(continues on next page)

(continued from previous page)

```
↪.dump files (lower precision), currently only supports orthogonal unit cells
)
```

Executing the code will generate trajectory files in .xyz and .dump formats, which can be opened with OVITO. For more details on structure file conversion, refer to *structure structure conversion*

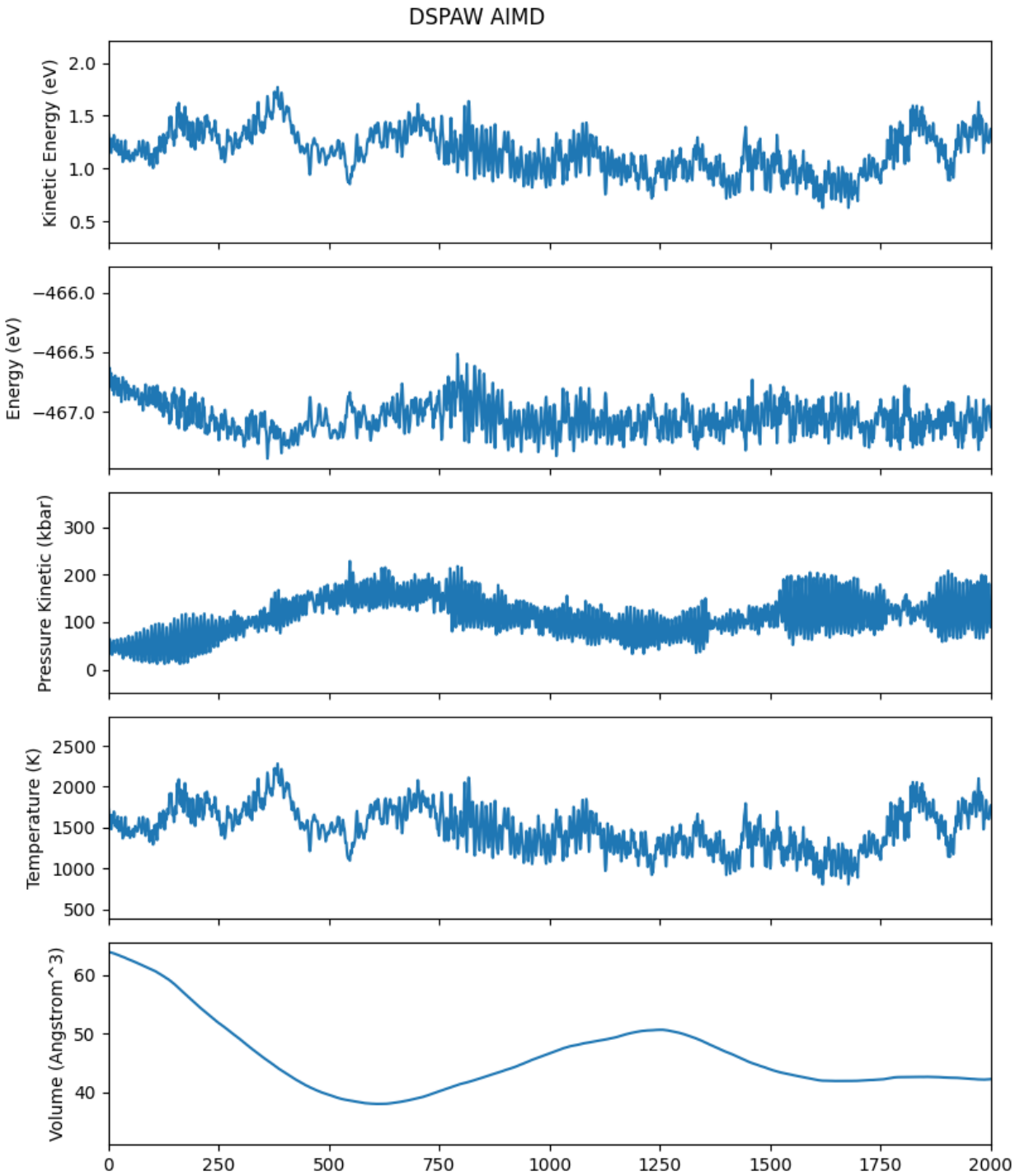
> **Note**
>
> OVITO and dspawpy do not support saving systems with non-orthogonal unit cells as dump files.

### 8.10.2 Changes in energy, temperature, etc. curves during the dynamics process.

- Refer to `10check_aimd_conv.py`:

```
# coding:utf-8
from dspawpy.plot import plot_aimd

plot_aimd(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5",  # Data file path
    show=False,  # Whether to pop up the image window
    figname="dspawpy_proj/dspawpy_tests/outputs/us/10aimd.png",  # Output image␣
↪file name
    flags_str="1 2 3 4 5",  # Select data types
)
# The meaning of flags_str is as follows
# 1. Kinetic energy
# 2. Total Energy
# 3. Pressure
# 4. Temperature
# 5. Volume
```

Executing the code will generate the following combined graph:




**Warning**

If you execute the above script by SSH connection to a remote server and encounter QT-related error messages, its possible that the program you are using (such as MobaXterm) is incompatible with the QT libraries. You can either switch programs (for example, VSCode or the systems built-in terminal command line), or add the following code starting from the second line of the Python script:

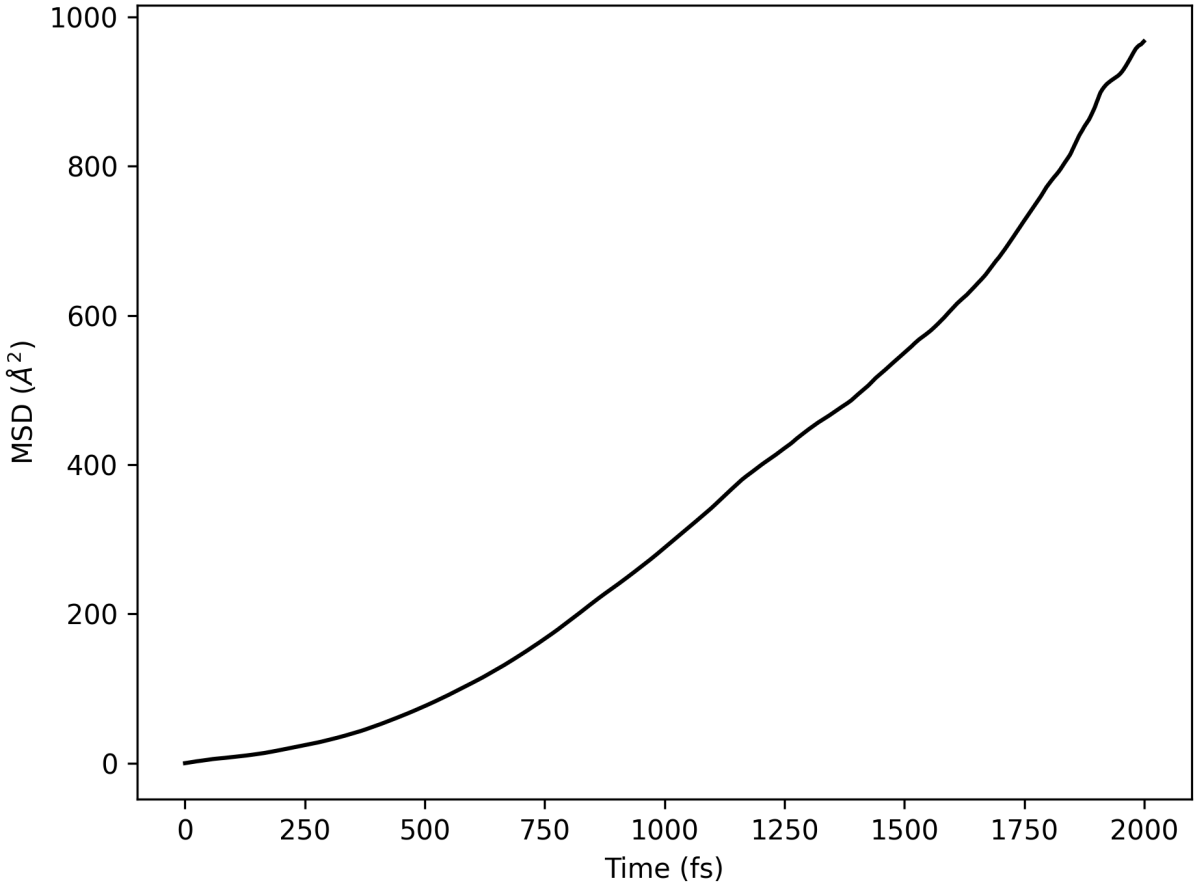
```
import matplotlib
matplotlib.use('agg')
```

### 8.10.3 Mean Squared Displacement (MSD) Analysis

- See 10aimd_msd.py:

```
# coding:utf-8
from dspawpy.analysis.aimdtools import get_lagtime_msd, plot_msd

# If AIMD is not completed in one go, you can assign multiple h5 file paths to the␣
↪datafile parameter in a list form
lagtime, msd = get_lagtime_msd(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5",  # Data file path
    select="all",  # Default selects all atoms
    msd_type="xyz",  # Default to calculate MSD in the xyz directions
    timestep=None,  # Default reads the timestep from the datafile
)
# Plot the graph using the obtained data and save it
plot_msd(
    lagtime,  # X-axis coordinate
    msd,  # vertical axis
    xlim=None,  # Set the display range of the x-axis
    ylim=None,  # Set the display range of the y-axis
    figname="dspawpy_proj/dspawpy_tests/outputs/us/10MSD.png",  # Output image␣
↪filename
    show=False,  # Whether to pop up the image window
    ax=None,  # Optional subplot specification
)
```

Executing the code will generate an image similar to the following:




**Warning**

If you execute the above script by SSH connection to a remote server and encounter QT-related error messages, it might be due to incompatibility between the program youre using (like MobaXterm, etc.) and the QT libraries. Either switch to a different program (such as VSCode or the systems built-in terminal), or add the following code starting from the second line of the Python script:

```
import matplotlib
matplotlib.use('agg')
```

### 8.10.4 Root Mean Square Deviation (RMSD) Analysis

- See `10aimd_rmsd.py`:

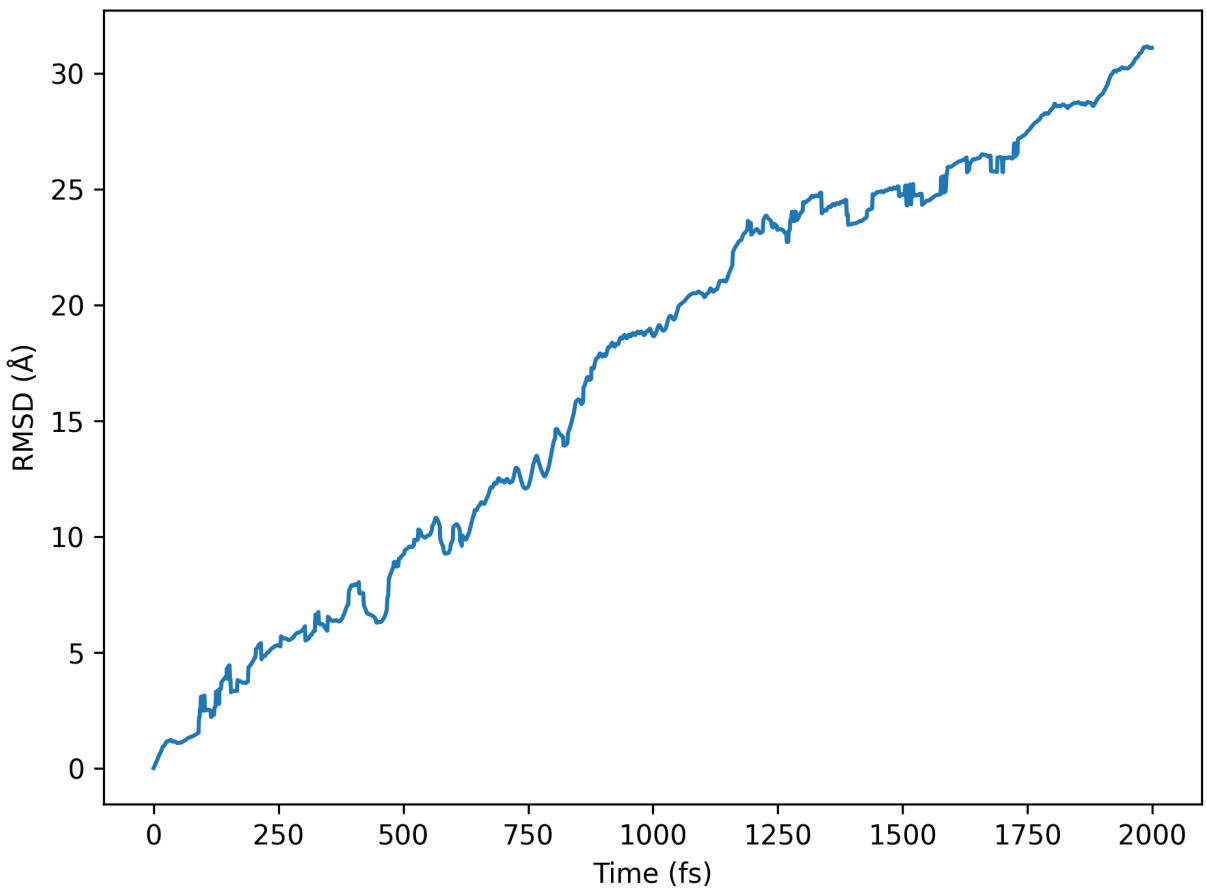
```
# coding:utf-8
from dspawpy.analysis.aimdtools import get_lagtime_rmsd, plot_rmsd

# If AIMD is not completed in a single run, you can assign multiple paths of h5␣
↪files in list form to the datafile parameter
lagtime, rmsd = get_lagtime_rmsd(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5",
```

(continues on next page)

(continued from previous page)

```
    timestep=None,  # Data file path  # Default reads the time step from the datafile
)
plot_rmsd(
    lagtime,  # Horizontal coordinate
    rmsd,  # vertical coordinate
    xlim=None,  # Set the display range of the x-axis
    ylim=None,  # Set the display range of the y-axis
    figname="dspawpy_proj/dspawpy_tests/outputs/us/10RMSD.png",  # Output image filename
    show=False,  # Whether to pop up the image window
    ax=None,  # Optional subplot specification
)
```

Executing the code will generate an image similar to the following:




**Warning**

If you execute the above script by connecting to a remote server via SSH, and QT-related error messages appear, it may be due to incompatibility between the program used (e.g., MobaXterm) and the QT libraries. Either

change the program (such as VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

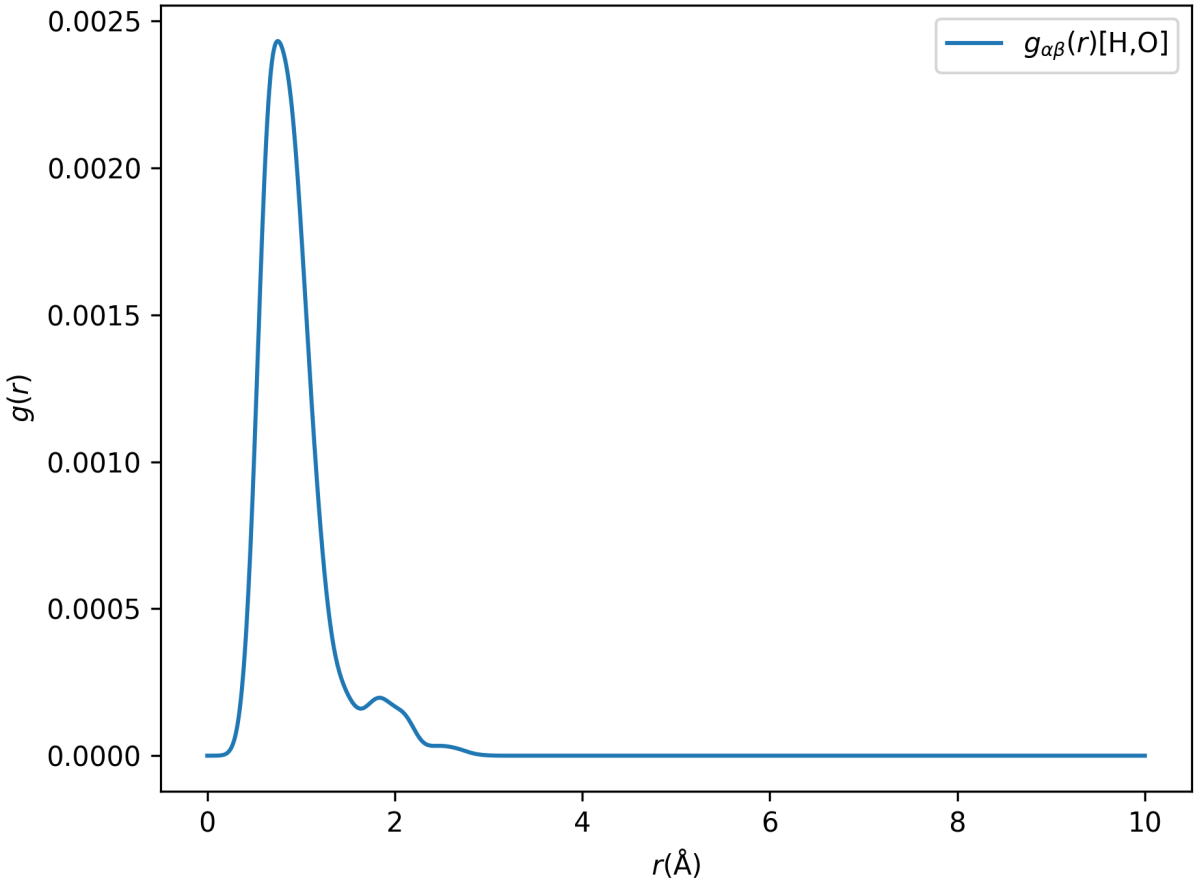
```
import matplotlib
matplotlib.use('agg')
```

### 8.10.5 Analysis of Atomic Pair Distribution Functions or Radial Distribution Functions (RDFs)

- See `10aimd_rdf.py` :

```
# coding:utf-8
from dspawpy.analysis.aimdtools import get_rs_rdfs, plot_rdf

# If AIMD is not completed in one go, you can assign multiple h5 file paths to the datafile parameter in the form of a list.
rs, rdfs = get_rs_rdfs(
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5",  # Data file path
    ele1="H",  # Central element
    ele2="O",  # Target element
    rmin=0.0,  # Minimum radius
    rmax=10.0,  # Maximum radius
    ngrid=1000,  # Number of grid points
    sigma=0.1,  # sigma value
)
plot_rdf(
    rs,  # x-axis values
    rdfs,  # Vertical coordinate
    "H",  # Central element
    "O",  # Object element
    figname="dspawpy_proj/dspawpy_tests/outputs/us/10RDF.png",  # Image save path
    show=False,  # Whether to pop up the image window
    ax=None,  # Subplot can be specified
)
```

Executing the code will generate an image similar to the following:




- The statistical calculations involved in this section are complex; please refer to the function API for more details.

### API: plot_aimd(), get_lagtime_msd(), plot_msd(), get_rs_rdfs(), plot_rdf(), get_lagtime_rmsd(), plot_rmsd()

- The `plot_aimd` function can be used to help check the convergence of key physical quantities during AIMD calculations:

`dspawpy.plot.`**`plot_aimd`**(*datafile: str = 'aimd.h5', show: bool = True, figname: str = 'aimd.png', flags_str: str = '12345', raw: bool = False*)

Plot the convergence process of key physical quantities after the AIMD task completion

aimd.h5 -> aimd.png

**Parameters**

- **`datafile`** – Location of the h5 file. For example, aimd.h5 or [aimd.h5, aimd2.h5]
- **`show`** – Whether to display the interactive interface. Default is False
- **`figname`** – Path to the saved image. Default aimd.h5
- **`flags_str`** – Subplot number. 1. Kinetic Energy 2. Total Energy 3. Pressure 4. Temperature 5. Volume
- **`raw`** – Whether to output plot data to a CSV file

**Returns**
Image path, default aimd.png

**Return type**
figname

### Examples

```
>>> from dspawpy.plot import plot_aimd
```

Read the contents of the aimd.h5 file, plot the convergence process graphs of kinetic energy, total energy, temperature, and volume, and save the corresponding data to rawaimd_*.csv.

```
>>> plot_aimd(datafile='dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5', flags_
↪str='1 2 3 4 5', raw=True, show=False, figname="dspawpy_proj/dspawpy_tests/
↪outputs/doctest/aimdconv.png")
>>> plot_aimd(datafile='dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.json',␣
↪flags_str='1 2 3 4 5', show=False, figname="dspawpy_proj/dspawpy_tests/
↪outputs/doctest/aimdconv_json.png")
```

- The `get_*` and `plot_*` functions are responsible for reading key physical quantities from the AIMD calculation process:

`dspawpy.analysis.aimdtools.`**`get_lagtime_msd`**(*datafile: str | List[str], select: str | List[int] = 'all', msd_type: str = 'xyz', timestep: float | None = None*)

Calculate the mean squared displacement at different time steps

**Parameters**

- **`datafile`** –
  - Path to *aimd.h5* or *aimd.json* files, or a directory containing these files (prioritizes searching for *aimd.h5*)
  - Written as a list, the data will be read sequentially and merged together
  - For example [aimd1.h5, aimd2.h5, /data/home/my_aimd_task]
- **`select`** – Select atomic number or element; atomic numbers start from 0; default is all, which calculates all atoms
- **`msd_type`** – Calculate the type of MSD, options: xyz, xy, xz, yz, x, y, z, default is xyz, which calculates all components
- **`timestep`** – Time interval between adjacent structures, in units of fs, default None, will be read from datafile, if failed, set to 1.0fs; If not None, this value will be used to calculate the time series

**Returns**

- **lagtime** (*np.ndarray*) – Time series
- **result** (*np.ndarray*) – Mean square displacement sequence

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_lagtime_msd
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.json', timestep=0.1)
Calculating MSD...
```

(continues on next page)

(continued from previous page)

```
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.h5')
Calculating MSD...
>>> lagtime
array([0.000e+00, 1.000e+00, 2.000e+00, ..., 1.997e+03, 1.998e+03,
       1.999e+03])
>>> msd
array([0.00000000e+00, 3.75844096e-03, 1.45298732e-02, ...,
       7.98518472e+02, 7.99267490e+02, 7.99992702e+02])
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.h5', select='H')
Calculating MSD...
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.json', select=[0,1])
Calculating MSD...
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.h5', select=['H','O'])
Calculating MSD...
>>> lagtime, msd = get_lagtime_msd(datafile='dspawpy_proj/dspawpy_tests/inputs/
↪2.18/aimd.json', select=0)
Calculating MSD...
```

`dspawpy.analysis.aimdtools.`**`get_lagtime_rmsd`**(*datafile: str | List[str], timestep: float | None = None*)

**Parameters**

- **`datafile`** –
  - Path to *aimd.h5* or *aimd.json* files, or a directory containing these files (prioritizes searching for *aimd.h5*).
  - Written as a list, the data will be read sequentially and merged together
  - For example [aimd1.h5, aimd2.h5, /data/home/my_aimd_task]
- **`timestep`** – Time interval between adjacent structures, in fs, default None, will be read from datafile, set to 1.0fs if failed; If not None, it will be used to calculate the time series

**Returns**

- **lagtime** (*numpy.ndarray*) – Time series
- **rmsd** (*numpy.ndarray*) – Root mean square deviation sequence

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_lagtime_rmsd
>>> lagtime, rmsd = get_lagtime_rmsd(datafile='dspawpy_proj/dspawpy_tests/
↪inputs/2.18/aimd.json')
Calculating RMSD...
>>> lagtime, rmsd = get_lagtime_rmsd(datafile='dspawpy_proj/dspawpy_tests/
↪inputs/2.18/aimd.h5', timestep=0.1)
Calculating RMSD...
>>> lagtime
array([0.000e+00, 1.000e-01, 2.000e-01, ..., 1.997e+02, 1.998e+02,
```

(continues on next page)

(continued from previous page)

```
       1.999e+02])
>>> rmsd
array([ 0.        ,  0.05321816,  0.09771622, ..., 28.27847679,
       28.28130893, 28.28414224])
```

`dspawpy.analysis.aimdtools.get_rs_rdfs`(*datafile: str | List[str], ele1: str, ele2: str, rmin: float = 0, rmax: float = 10, ngrid: int = 101, sigma: float = 0*)

Compute the radial distribution function (RDF).

**Parameters**

- **datafile** –
  - Path to *aimd.h5* or *aimd.json* files, or a directory containing these files (prioritizes searching for *aimd.h5*)
  - Written as a list, the data will be read sequentially and merged together
  - For example [aimd1.h5, aimd2.h5, /data/home/my_aimd_task]
- **ele1** – Central element
- **ele2** – Adjacent elements
- **rmin** – Radial distribution minimum value, default is 0
- **rmax** – Radial distribution maximum value, default is 10
- **ngrid** – Number of grid points in the radial distribution, default is 101
- **sigma** – Smoothing parameter

**Returns**

- **r** (*numpy.ndarray*) – Grid points for the radial distribution
- **rdf** (*numpy.ndarray*) – Radial distribution function

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_rs_rdfs
>>> rs, rdfs = get_rs_rdfs(datafile='dspawpy_proj/dspawpy_tests/inputs/2.18/
↪aimd.h5',ele1='H',ele2='O', sigma=1e-6)
Calculating RDF...
>>> rs, rdfs = get_rs_rdfs(datafile='dspawpy_proj/dspawpy_tests/inputs/2.18/
↪aimd.h5',ele1='H',ele2='O')
Calculating RDF...
>>> rdfs
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.00646866,
       0.01098199, 0.0004777 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

(continues on next page)

(continued from previous page)

```
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        ])
```

`dspawpy.analysis.aimdtools.`**`plot_msd`**(*lagtime*, *result*, *xlim: Sequence | None = None*, *ylim: Sequence | None = None*, *figname: str | None = None*, *show: bool = True*, *ax=None*, ***kwargs*)

Compute mean squared displacement (MSD) after the AIMD task is completed

**Parameters**

- **lagtime** (*np.ndarray*) – Time series
- **result** (*np.ndarray*) – Mean squared displacement sequence
- **xlim** – x-axis range, default None, set automatically
- **ylim** – y-axis range, default to None, automatically set
- **figname** – Image name, default to None, do not save the image
- **show** – Whether to display the image, default is True
- **ax** – Used to draw the image on a subplot in matplotlib
- ****kwargs** (*dict*) – Other parameters, such as line width, color, etc., are passed to plt.plot function

**Return type**
Image after MSD analysis

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_lagtime_msd, plot_msd
```

Specify the location of the h5 file, use the get_lagtime_msd function to obtain data, and the select parameter selects the nth atom (not by element)

```
>>> lagtime, msd = get_lagtime_msd('dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.
↪h5', select=[0])
Calculating MSD...
```

Plot the data and save the figure

```
>>> plot_msd(lagtime, msd, xlim=[0,800], ylim=[0,1000], figname='dspawpy_proj/
↪dspawpy_tests/outputs/doctest/MSD.png', show=False)
==> ...MSD.png
...
```

dspawpy.analysis.aimdtools.**plot_rdf**(*rs*, *rdfs*, *ele1: str*, *ele2: str*, *xlim: Sequence | None = None*, *ylim: Sequence | None = None*, *figname: str | None = None*, *show: bool = True*, *ax=None*, ***kwargs*)

Post-AIMD analysis of rdf and plotting.

**Parameters**

- **rs** (*numpy.ndarray*) – Radial distribution grid points
- **rdfs** (*numpy.ndarray*) – Radial distribution function
- **ele1** – Center element
- **ele2** – Adjacent elements
- **xlim** – x-axis range, default to None, i.e., set automatically
- **ylim** – y-axis range, default to None, i.e., automatically set
- **figname** – Image name, default to None, meaning no image is saved
- **show** – Whether to display the image, default to True
- **ax** (*matplotlib.axes.Axes*) – Axis for plotting, default is None, which means creating a new axis
- ****kwargs** (*dict*) – Other parameters, such as line width, color, etc., are passed to plt.plot function

**Return type**

Image after RDF analysis

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_rs_rdfs, plot_rdf
```

First obtain the rs and rdfs data as the x and y axis data

```
>>> rs, rdfs = get_rs_rdfs('dspawpy_proj/dspawpy_tests/inputs/2.18/aimd.h5', 'H
↪', 'O', rmax=6)
Calculating RDF...
```

Passing x and y data to the plot_rdf function to plot

```
>>> plot_rdf(rs, rdfs, 'H','O', xlim=[0, 6], ylim=[0, 0.015],figname='dspawpy_
↪proj/dspawpy_tests/outputs/doctest/RDF.png', show=False)
==> ...RDF.png
```

dspawpy.analysis.aimdtools.**plot_rmsd**(*lagtime*, *result*, *xlim: Sequence | None = None*, *ylim: Sequence | None = None*, *figname: str | None = None*, *show: bool = True*, *ax=None*, ***kwargs*)

Post-AIMD analysis of RMSD and plotting

**Parameters**

- **lagtime** – Time series
- **result** – Root mean square deviation sequence
- **xlim** – x-axis range
- **ylim** – y-axis range

- **figname** – Image save path
- **show** – Whether to display the image
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – If plotting subplots, pass the subplot object
- ****kwargs** (*dict*) – Parameters passed to plt.plot

**Return type**
Image of RMSD analysis of structures

### Examples

```
>>> from dspawpy.analysis.aimdtools import get_lagtime_rmsd, plot_rmsd
```

*timestep* represents the time step length

```
>>> lagtime, rmsd = get_lagtime_rmsd(datafile='dspawpy_proj/dspawpy_tests/
↪inputs/2.18/aimd.h5', timestep=0.1)
Calculating RMSD...
>>> lagtime, rmsd = get_lagtime_rmsd(datafile='dspawpy_proj/dspawpy_tests/
↪inputs/2.18/aimd.json', timestep=0.1)
Calculating RMSD...
```

Saving directly as RMSD.png image

```
>>> plot_rmsd(lagtime, rmsd, xlim=[0,200], ylim=[0, 30],figname='dspawpy_proj/
↪dspawpy_tests/outputs/doctest/RMSD.png', show=False)
==> ...RMSD.png
...
```

- For manual data extraction and processing, refer to:

```
from dspawpy.io.read import get_sinfo
from dspawpy.io.structure import read


aimd_h5_files = ['aimd1.h5','aimd2.h5','aimd3.h5'] # Extract and merge data␣
↪sequentially from multiple completed aimd.h5 files

# Read data from multiple aimd.h5 files at once and create a list of pymatgen␣
↪Structures
pymatgen_structures = read(datafile=aimd_h5_files)

# Or extract the arrays
for i, df in enumerate(aimd_h5_files): # Get data from each aimd.h5 file␣
↪sequentially
    Nstep, elements, positions, lattices, D_mag_fix = get_sinfo(df)
```

**Warning**

If you connect to a remote server via SSH and execute the above script, and you encounter QT-related error messages, its possible that the program youre using (such as MobaXterm) is incompatible with the QT libraries. Ei-

ther change the program (for example, VSCode or the systems built-in terminal command line), or add the following code, starting on the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

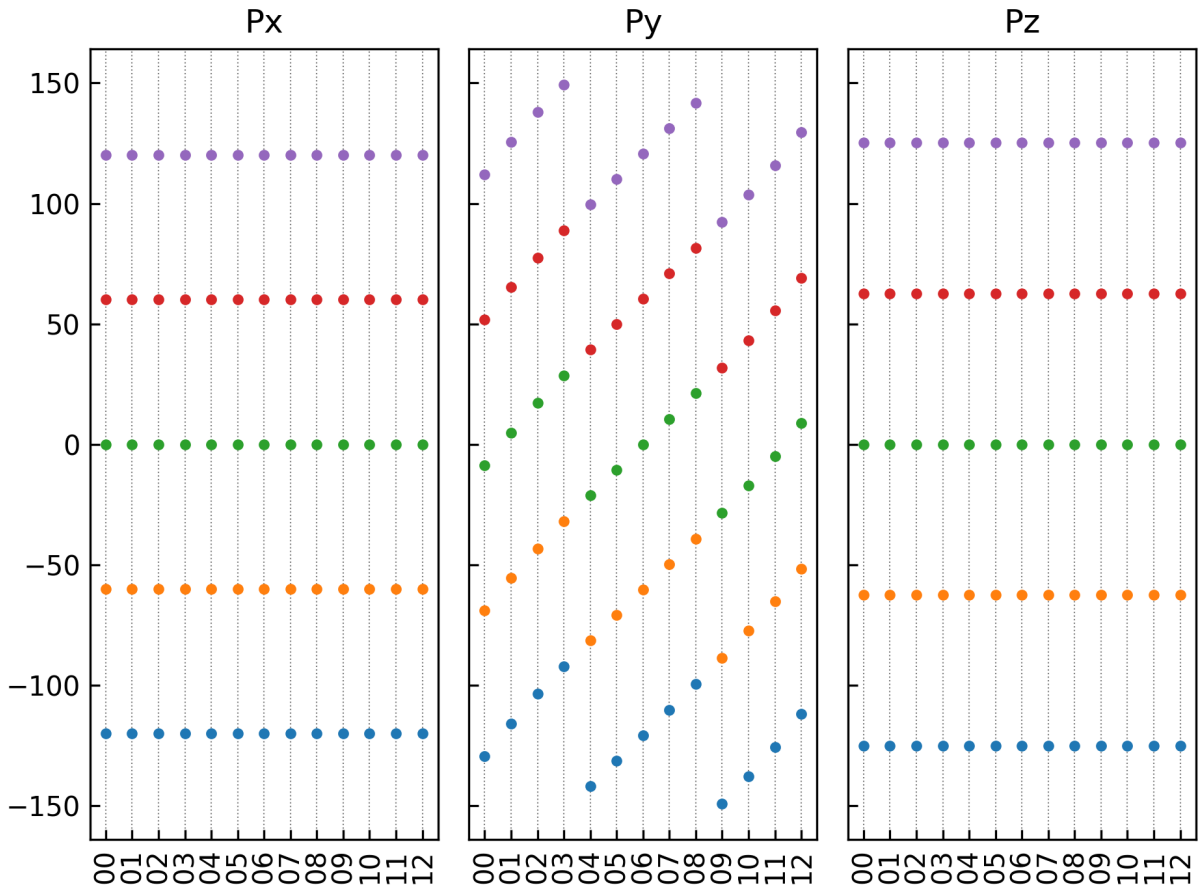
## 8.11 Ferroelectric Polarization Data Processing

For a quick start, take the series of *scf.h5* files obtained from ferroelectric calculations on the $HfO_2$ system as an example:

- See `11Ferri.py`:

```
# coding:utf-8
from dspawpy.plot import plot_polarization_figure

plot_polarization_figure(
    directory="dspawpy_proj/dspawpy_tests/inputs/2.20",  # Path for iron␣
↪polarization calculation
    repetition=2,  # Number of times to repeat the data points when plotting
    figname="dspawpy_proj/dspawpy_tests/outputs/us/11pol.png",  # Output␣
↪polarization figure filename
    show=False,  # Whether to display the polarization figure
)  # --> pol.png
```

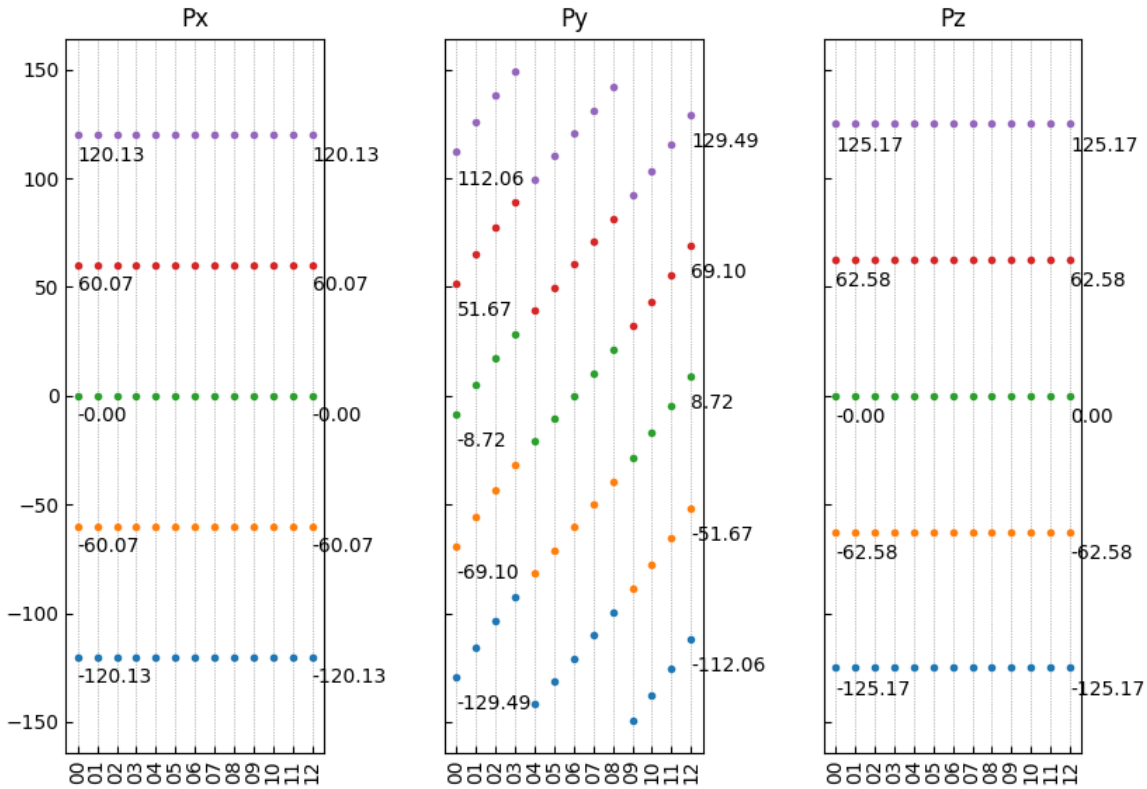
Executing the code will generate the following combined figure:




The ferroelectric polarization values for the head and tail configurations can be found below:

```
from dspawpy.plot import plot_polarization_figure

python
plot_polarization_figure(directory='.', annotation=True, annotation_style=1)  # Displays the ferroelectric polarization values for the initial and final configurations.
```

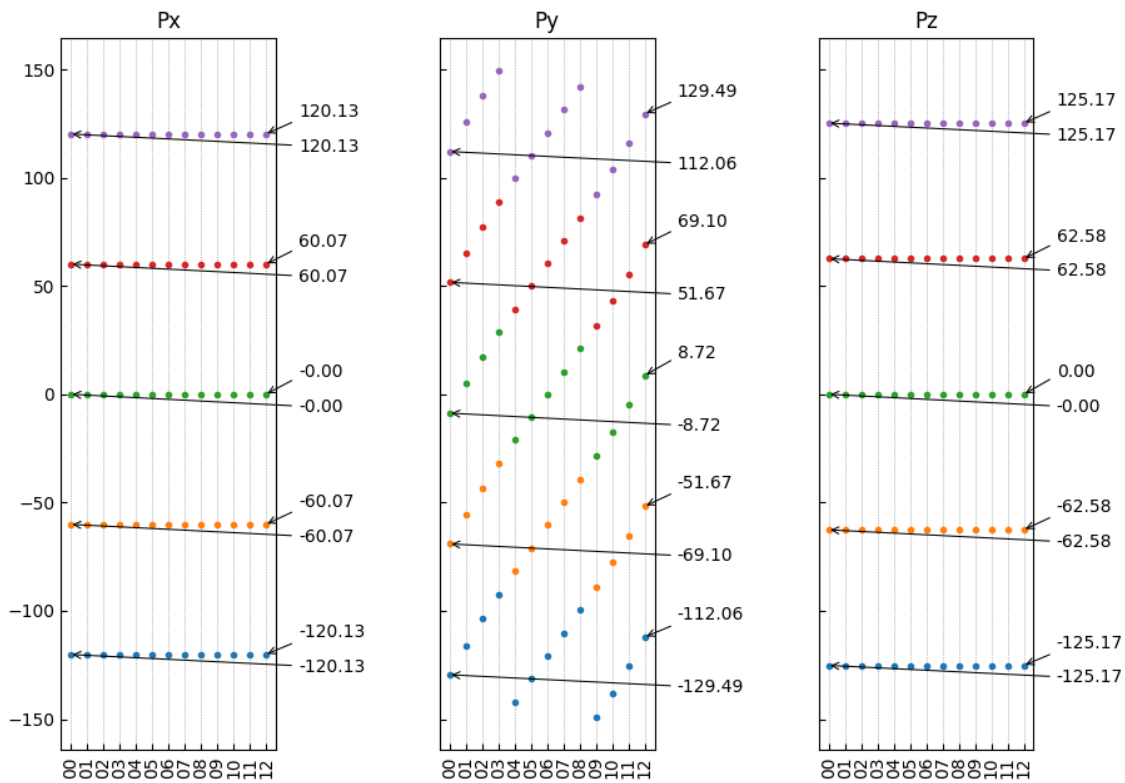
The code will generate the following combined plot:




Alternatively, a second annotation style can be used:

```
from dspawpy.plot import plot_polarization_figure

plot_polarization_figure(directory='.', annotation=True, annotation_style=2)  #␣
↪Displays the ferroelectric polarization values for the initial and final␣
↪configurations.
```

The code will generate the following combined plot:




### API: plot_polarization_figure()

- The *plot_polarization_figure* function is responsible for plotting the ferroelectric polarization figure:

`dspawpy.plot.`**`plot_polarization_figure`**(*directory: str*, *repetition: int = 2*, *annotation: bool = False*, *annotation_style: int = 1*, *show: bool = True*, *figname: str = 'pol.png'*, *raw: bool = False*)

Plot the polarization results of the iron electrode

**Parameters**

- **`directory`** – Main directory for the iron polarization calculation task
- **`repetition`** – Number of times to repeat drawing along the upper (or lower) direction, default 2
- **`annotation`** – Whether to display the polarization values of the iron electrodes at the beginning and end configurations, displayed by default
- **`show`** – Interactive display of the image, default True
- **`figname`** – Image save path, default pol.png
- **`raw`** – Whether to save the raw data to a CSV file

**Returns**

**axes** – Can be passed to other functions for further processing

**Return type**

matplotlib.axes._subplots.AxesSubplot

### Examples

```
>>> from dspawpy.plot import plot_polarization_figure
>>> result = plot_polarization_figure(directory='dspawpy_proj/dspawpy_tests/
↪inputs/2.20', figname='dspawpy_proj/dspawpy_tests/outputs/doctest/pol1.png',␣
↪show=False, annotation=True, annotation_style=1)
>>> result = plot_polarization_figure(directory='dspawpy_proj/dspawpy_tests/
↪inputs/2.20', figname='dspawpy_proj/dspawpy_tests/outputs/doctest/pol2.png',␣
↪show=False, annotation=True, annotation_style=2)
```

**Warning**

If you encounter QT-related error messages when executing the above script via SSH connection to a remote server, it may be due to incompatibility between the program used (e.g., MobaXterm) and the QT library. Either change the program (e.g., VSCode or the systems built-in terminal command line), or add the following code starting from the second line of your Python script:

```
import matplotlib
matplotlib.use('agg')
```

## 8.12 Zero-Point Vibrational Energy Data Processing

Taking the *frequency.txt* file obtained from a quick start $CO$ system frequency calculation as an example, the zero-point vibrational energy is calculated based on the following formula:

$$ZPE = \sum_{i=1}^{3N} \frac{h\nu_i}{2}$$

where $\nu_i$ are the normal mode frequencies, $h$ is Plancks constant ($6.626 \times 10^{-34} J \cdot s$), and $N$ is the number of atoms.

- See `12getZPE.py`:

```
# coding:utf-8
from dspawpy.io.utils import getZPE

# Import the frequency.txt file obtained from frequency calculation
getZPE(
    fretxt="dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt",
)
```

The code execution results will be saved to the *ZPE.dat* file, and the file content is as follows:

```
Data read from D:\quickstart\2.13\frequency.txt
Frequency (meV)
284.840038

--> Zero-point energy,  ZPE (eV): 0.142420019
```

### API: getZPE()

- The `getZPE` function is responsible for calculating the zero-point vibrational energy:

  Some functions are extracted from [ase](https://wiki.fysik.dtu.dk/ase/index.html).

  `dspawpy.io.utils.getZPE`(*fretxt: str = 'frequency.txt'*)

  Read data from fretxt, calculate ZPE

  The results will also be saved to ZPE_TS.dat.

  **Parameters**
  **fretxt** – Path to the file recording frequency information, default to frequency.txt in the current path

  **Returns**
  Zero-point energy

  **Return type**
  ZPE

#### Examples

```
>>> from dspawpy.io.utils import getZPE
>>> ZPE=getZPE(fretxt='dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt')
--> Zero-point energy,  ZPE (eV): 0.1424200165
```

## 8.13 TS Hot Calibration Energy

### 8.13.1 Contribution of the entropy change of the adsorbate to the energy

Calculation is based on the following formula:

$$\Delta S_{ads}\left(0 \rightarrow T, P^{0}\right)=S_{vib}=\sum_{i=1}^{3N}\left[\frac{N_{\mathrm{A}} h \nu_{i}}{T\left(e^{h \nu_{i} / k_{\mathrm{B}} T}-1\right)}-R \ln \left(1-e^{-h \nu_{i} / k_{\mathrm{B}} T}\right)\right]$$

Here, $\Delta S_{ads}$ represents the entropy change of the adsorbate, calculated based on the harmonic approximation. $S_{vib}$ represents the vibrational entropy. $\nu_i$ is the normal mode frequency, $N_A$ is Avogadros constant ($6.022 \times 10^{23} mol^{-1}$), $h$ is Plancks constant ($6.626 \times 10^{-34} J \cdot s$), $k_B$ is the Boltzmann constant ($1.38 \times 10^{-23} J \cdot K^{-1}$), $R$ is the ideal gas constant ($8.314 J \cdot mol^{-1} \cdot K^{-1}$), and $T$ is the system temperature in units of $K$.

- See `13getTSads.py` for reference:

```
# coding:utf-8
from dspawpy.io.utils import getTSads

# Import the frequency.txt file calculated from frequency, temperature can be␣
↪modified arbitrarily
getTSads(
    fretxt="dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt",
    T=298.15,
)
```

The execution result will be saved to the *TS.dat* file, with the following content:

```
Data read from D:\quickstart\2.13\frequency.txt
Frequency (THz)
68.873994

--> Entropy contribution,  T*S (eV): 4.7566990201851275e-06
```

### 8.13.2 Ideal gas entropy contribution to energy

Calculations are based on the following formula:

$$
\begin{aligned}
S(T,P) &= S\left(T,P^{\circ}\right) - k_{\mathrm{B}} \ln \frac{P}{P^{\circ}} \\
&= S_{\text{trans}} + S_{\text{rot}} + S_{\text{elec}} + S_{\text{vib}} - k_{\mathrm{B}} \ln \frac{P}{P^{\circ}}
\end{aligned}
$$

Where:

$$
S_{\text{trans}} = k_{\mathrm{B}} \left\{ \ln \left[ \left( \frac{2\pi M k_{\mathrm{B}} T}{h^2} \right)^{3/2} \frac{k_{\mathrm{B}} T}{P^{\circ}} \right] + \frac{5}{2} \right\}
$$

$$
S_{\text{rot}} = \begin{cases} 0 & \text{, monatomic} \\ k_{\mathrm{B}} \left[ \ln \left( \frac{8\pi^2 I k_{\mathrm{B}} T}{\sigma h^2} \right) + 1 \right] & \text{, linear} \\ k_{\mathrm{B}} \left\{ \ln \left[ \frac{\sqrt{\pi I_A I_{\mathrm{B}} I_{\mathrm{C}}}}{\sigma} \left( \frac{8\pi^2 k_B T}{h^2} \right)^{3/2} \right] + \frac{3}{2} \right\} & \text{, nonlinear} \end{cases}
$$

$$
S_{\text{elec}} = k_{\mathrm{B}} \ln[2 \times (\text{ total spin }) + 1]
$$

$$
S_{\text{vib}} = k_{\mathrm{B}} \sum_{i}^{\text{vib DOF}} \left[ \frac{\epsilon_i}{k_{\mathrm{B}} T \left( e^{\epsilon_i / k_{\mathrm{B}} T} - 1 \right)} - \ln \left( 1 - e^{-\epsilon_i / k_{\mathrm{B}} T} \right) \right]
$$

where $I_A$ to $I_C$ are the three principal moments of inertia for a non-linear molecule, $I$ is the degenerate moment of inertia for a linear molecule, and $\sigma$ is the symmetry number of the molecule. Furthermore, monatomic refers to a monatomic molecule, linear refers to a linear molecule, and nonlinear refers to a non-linear molecule. total spin is the total spin quantum number. vib DOF represents vibrational degrees of freedom.

- Refer to the `13getTSgas.py` script for processing:

```python
# coding:utf-8
from dspawpy.io.utils import getTSgas

# Read elements and coordinates from the calculation result file (json or h5)
TSgas = getTSgas(
    fretxt="dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt",
    datafile="dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.h5",
    potentialenergy=-0.0,
    geometry="linear",
    symmetrynumber=1,
    spin=1,
    temperature=298.15,
    pressure=101325.0,
)
print("Entropy contribution, T*S (eV)", TSgas)

# If only the frequency.txt file is available, the calculation can be completed by␣
```

(continues on next page)

(continued from previous page)

```
↪manually specifying the elements and coordinates
# TSgas = getTSgas(fretxt='dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt',␣
↪datafile=None, potentialenergy=-0.0, elements=['H', 'H'], geometry='linear',␣
↪positions=[[0.0, 0.0, 0.0], [0.0, 0.0, 1.0]], symmetrynumber=1, spin=1,␣
↪temperature=298.15, pressure=101325.0)
```

### API: getTSads(), getTSgas()

- The `getTSads` function is responsible for calculating the contribution of adsorbate entropy change to the energy:

  Some functions are extracted from [ase](https://wiki.fysik.dtu.dk/ase/index.html).

  `dspawpy.io.utils.`**`getTSads`**(*fretxt: str = 'frequency.txt'*, *T: float = 298.15*)

  Read data from fretxt, calculate ZPE and TS

  Will also save the results to TSads.dat

  **Parameters**

  - **fretxt** – Path to the file recording frequency information, default frequency.txt in the current path
  - **T** – Temperature, unit K, default 298.15

  **Returns**
  Entropy correction

  **Return type**
  TS

  #### Examples

  ```
  >>> from dspawpy.io.utils import getTSads
  >>> TSads=getTSads(fretxt='dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt
  ↪', T=298.15)
  --> T*S (eV): 4.7566997225177686e-06
  ```

- The `getTSgas` function is responsible for calculating the contribution of ideal gas entropy change to energy:

  Some functions are extracted from [ase](https://wiki.fysik.dtu.dk/ase/index.html).

  `dspawpy.io.utils.`**`getTSgas`**(*fretxt='frequency.txt'*, *datafile='.'*, *potentialenergy: float = 0.0*, *elements=None*, *geometry='linear'*, *positions=None*, *symmetrynumber=1*, *spin=1*, *temperature=298.15*, *pressure: float = 101325*, *verbose: bool = False*)

  Energy contribution to entropy under the ideal gas approximation

  **Parameters**

  - **fretxt** – Path to the file recording frequency information, default is frequency.txt in the current path
  - **datafile** – Path to the JSON or h5 file or folder containing them, default to the current path; If set to None, the elements and positions parameters must be provided
  - **potentialenergy** – Potential energy, unit eV
  - **elements** – List of elements, if
  - **geometry** – Molecular geometry, monatomic, linear, nonlinear

- **positions** – Atomic coordinates, unit Angstrom
- **symmetrynumber** – Symmetry number
- **spin** – Spin number
- **temperature** – Temperature, unit K
- **pressure** – Pressure, unit Pa

**Returns**
Under the ideal gas approximation, calculates the energy contribution to entropy, in units of eV

**Return type**
TSgas

### Examples

```
>>> from dspawpy.io.utils import getTSgas
>>> TSgas=getTSgas(fretxt='dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.txt
↪', datafile='dspawpy_proj/dspawpy_tests/inputs/2.13/frequency.h5',␣
↪potentialenergy=-0.0,  geometry='linear', symmetrynumber=1, spin=1,␣
↪temperature=298.15, pressure=101325.0)
--> T*S (eV): 0.8515317035550232
```

## 8.14 Appendix

- Quickly download all scripts by clicking `UserScripts.zip`
- dspawpy Changelog

# 9

# Frequently Asked Questions (FAQ)

## 9.1 Common License Error Messages

- Error message: `Error code: -10, Get License File Error`
- Error details: *License file not found or insufficient permissions to open it.*

---

- Error message: `Error code: -20, Get License Product Error`
- Error details: *Failed to get product information*

---

- Error message: `Error code: -30, Check Local Environment Error`
- Error details: *Local hardware information verification error*

---

- Error message: `Error code: -40, Check Install Path Error`
- Error details: *Local install path verification error*

---

- Error Message: `Error code: -50, Check Validate White User Error`
- Error details: *Whitelist validation error, the current user is not in the whitelist.*

---

- Error message: `Error code: -60, Check Device Studio license Error`
- Error Details: *Incorrect DS product information*

---

- Error message: `Error code: -70, Check Device Studio license Error`

- Error details: *DS is not able to use the DS-PAW software from the product catalog*

---

- Error message: `Error code:  -80, Check Device Studio license Error`
- Error details: *DS-PAWs current version in DS license is higher than the registered version*

---

- Error message: `Error code:  -90, Check Device Studio license Error`
- Error Details: *DS-PAW in the DS license has expired. Registration validity*

## 9.2 Inputcheck: Common Error Messages for Input Files

- Error message: `Parameters task error`
- Error details: *Incorrect task parameter name or parameter setting*

---

- Error Message: `Parameters Check error`
- Error Details: *Parameter Name Error*

---

- Error Message: `Parameters type error`
- Error details: *Parameter type setting error*

---

- Error Details: `Parameters data error`
- Error details: *Issue with optional parameter value settings*

---

- Error message: *Parameters size error*
- Error details: *Issue with parameter size dimensions*

---

- Error message: *Parameters range error*
- Error details: *Parameter range error*

---

- Error message: *Structure key error*
- Error details: *Missing key in structure file*

---

- Error message: *Structure type error*
- Error details: *Keyword settings are incorrect in the structure file*

---

- Error message: *Structure size error*
- Error details: *Incorrect data size in the structure file*

## 9.3 Common Error Messages During Calculation

- Error Codes: `E1015/E1011/E1012/E1014/E1005`
- Error Details: *Error reading K-points*
- Solution: *Increase the k-point density in all directions (try increasing by about 20%, but do not increase the k-points corresponding to the vacuum direction) or modify cal.smearing and cal.sigma, e.g., set cal.smearing = 1, cal.sigma = 0.05*

---

- Error code: `E1188`
- Error details: *More than 4 k-points are required when using the tetrahedron method*
- Solution: *Increase the k-point density in each direction (try increasing by about 20%, no need to increase k-points in the vacuum direction) or modify cal.smearing and cal.sigma, e.g., set cal.smearing = 1, cal.sigma = 0.05*

---

- Error code: `E1005`
- Error details: *k-point shift read error*
- Solution: *Try using cal.ksampling= G*

---

- Error Message: `E1013`
- Error Details: *K-point path read error*
- Solution: *Try using cal.ksampling= G*

---

- Error Message: `E1022`
- Error details: *Error occurred when reading eigenvalues from wave.bin*
- Solution: *Adjust the input parameters of the two calculations to obtain the correct wave.bin*

---

- Error message: `E1024`
- Error details: *The grid size generated by the current calculation is inconsistent with that read from rho.bin*
- Solution: *Adjust the input parameters for both calculations to obtain the correct rho.bin*

---

- Error Message: `E1042/E1041`
- Error details: *ZBRENT algorithm encountered an error while searching for the root function*
- Solution: *Read the structure from the log file before the error, generate a new structure file, then increase the convergence accuracy with scf.convergence to continue the calculation; or modify the relaxation algorithm to relax.methods = QN and recalculate*

---

- Error message: `E1063`
- Error details: *An error occurred when executing the LAPACKE_zhegv_work function while using the davidson block method*

- Solution: *Adjust cal.methods*

---

- Error message: `E1064`
- Error Details: *An error occurred during the LAPACKE_zhegv_work function execution during diagonalization*
- Solution: *Adjust cal.methods*

---

- Error message: `E1073`
- Error details: *Error occurred during parallel acceleration*
- Solution: *Disable the -pob command in the submission script and resubmit the job.*

---

- Error Message: `E1115`
- Error details: *Lattice volume is zero*

---

- Error message: `E1186`
- Error details: *An error occurred while inverting the rotation matrix*
- Solution: *Turn off symmetry sys.symmetry = false*

---

- Error Message: `E1187`
- Error details: *Error occurred while inverting the rotation matrix*
- Solution: *Try using cal.ksamping= MP*

---

- Error message: `E1226`
- Error Details: *Error occurred during expansion*
- Solution: *Check and modify the structure file*

---

- Error message: `E1248`
- Error Details: *An error occurred in the LAPACKE_zpotrf_work function during the orthogonalization of the wave function.*
- Solution: *Set sys.symmetry = false and reduce relax.stepRange*

---

- Error message: `E1249`
- Error details: *An error occurred in the LAPACKE _ztrtri _work function during the orthogonalization of wave functions.*

---

- Error message: `E2024/E2025`
- Error details: *An error occurred when inverting the rotation matrix*

- Solution: *Improve the accuracy of symmetry judgment, such as setting sys.symmetryAccuracy = 1.0e-6*

---

- Error message: `E3058`
- Error Details: *Pseudopotential reading error*
- Solution: *DS-PAW currently provides 72 element pseudopotentials and does not support calculations with elements outside of the pseudopotential library; if the calculation system contains custom element names, you need to copy the corresponding files from the pseudopotential library to the calculation directory and rename them*

---

- Error Message: `E4001`
- Error Details: *Mismatch between the number of initial projection orbitals and Wannier functions in the Wannier calculation*
- Solution: *Adjust the number of initial projection orbitals in the structure.as file, or modify the parameter wannier.functions in the input.in file to make the two numbers consistent.*

---

- Error message: `E4024`
- Error Details: *Incorrect freezing window settings for Wannier calculation*
- Solution: *The number of bands within the frozen window must not exceed the number of Wannier functions. Reduce the frozen window.*

---

- Error message: `Failed to converge the scf calculation`
- Error message: *Electronic steps did not converge within the set number of steps.*
- Solution: *Try modifying the algorithm to cal.methods = 1, or increase cal.totalBands.*

## 9.4 Version FAQs

` 1. Compatibility Issues between DS-PAW and Device Studio: `

- Why cant the band structure and phonon spectra generated by DS-PAW 2023A be opened in Device Studio?

  DS-PAW 2023A changed Band to BandEnergies in output files to better reflect the physical meaning of the data, based on user suggestions. Compatibility has been implemented in the updated Device Studio 2022B-2.0.6 version. Alternatively, you can rename BandEnergies back to Band in the output file to allow it to be opened in versions of Device Studio prior to 2022B-2.0.6.

- Why cant the NEB data generated by DS-PAW 2023A be opened in DS?

  DS-PAW 2023A has adjusted the output files based on user suggestions. This includes unifying labels in neb0N.json/neb0N.h5 and neb.json/neb.h5, and adjusting the data structure to make the physical meaning of the data clearer. To ensure compatibility with the current version of Device Studio, we provide several neb processing scripts to meet various needs. For example, the neb_visualize.py script can be used to view any structure during the neb optimization process, convert the final neb configuration into an xyz trajectory file. The neb_check_results.py script can print the energy and force tables for each configuration in the NEB calculation, plot the energy barrier, and plot the energy and force convergence graphs for each image. For detailed usage instructions, please refer to the transition state data processing section in *Auxiliary Tool User Guide*. The 2023A version of Device Studio has been updated for compatibility. Please update Device Studio if you are unable to open the files.

2. Why is hybrid functional calculation no longer supported for *task=band* in DS-PAW 2023A?

   Due to the special nature of hybrid functionals, the actual calculation process for band structure calculations using *task=band* and *io.band=true* is identical. To avoid user confusion regarding the difference between the two, we no longer support hybrid functional calculations with *task=band* (non-self-consistent calculations).

## 9.5 Manual Related Issues

1. EPUB and MOBI ebooks display formatting errors, images appear out of place, and navigation links are incorrect.

This is likely a rendering issue with your reader. On Windows, try using Calibre or Sigil; on iOS, use the built-in Books app.

2. Math formulas not rendered in web browsers

   This is likely a network issue; please wait patiently for rendering to complete.

# 10

# Release Notes

## 10.1 2025A

### 10.1.1 Pseudopotential Update

LDA and PBE pseudopotentials for elements of periods 4-6 (K Ca Sc Ti V Fe Co Ni Cu Zn Ga Ge As Se Br Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te Hf Ta W Re Os Ir Pt Au Hg Pb Bi Po) have been updated to version 1.1, improving computational accuracy and lowering the cutoff energy.

### 10.1.2 Functional Improvements

1. Optimized memory consumption during pcharge calculation
2. Comprehensive code optimization significantly improves computation speed.

## 10.2 Release Notes for Version 2023A

### 10.2.1 New Features

1. Support for constant potential method in SCF calculations
2. Implicit solvent model is supported.
3. AIMD calculations now include a Langevin thermostat (barostat), adding support for NPT/NPH ensembles; *aimd.thermostat=none* is renamed to SA (simulated annealing).
4. Support for fitting and band interpolation calculations with Maximally Localized Wannier Functions (MLWF).

---

### 10.2.2 Pseudopotential Updates

LDA and PBE pseudopotentials for the first three periods (H, He, Li, Be, B, C, N, O, F, Ne, Na, Mg, Al, Si, P, S, Cl, Ar) are released in version 1.1, improving computational accuracy and reducing the cutoff energy.

---

### 10.2.3 IO Tuning

1. Added HDF5 format files as the default output file format for DS-PAW, and the JSON format output files will no longer be maintained.
2. Modified the output for the DS-PAW.log parameters.
3. Remove tmp folder

---

### 10.2.4 Feature Optimization

1. Added Pulay option for scf.mixType
2. Added atom and shape options to *relax.freedom*
3. Added the *relax.pressure* parameter.
4. Added parameters related to FFT grid: cal.FFTGrid, cal.supGrid
5. Added support for the *cal.opticalGrid* parameter when *io.optical=true*.
6. Added support for alternative writing styles in band.kpointsNumber
7. Added the *corr.dftuForm* parameter to determine the DFT+U method type.
8. Added support for exchange-correlation functionals compatible with semi-empirical VDW corrections.

---

Updated 2023A 2024/04/03

1. Supplement the output of the wave function derivative with respect to k during optical calculations.

#### 10.2.4.1 Feature optimization

1. Added *sys.spinDiff* parameter to restrict the difference in the number of spin-up and spin-down electrons.
2. Added *corr.coreEnergy* parameter to control whether core electron energy levels are calculated.
3. Fixed a bug where the *mag* parameter was read incorrectly in some cases.

### 10.2.5 Updated on 2024/03/15 (2023A)

#### 10.2.5.1 Feature Optimization

1. Fixed the issue where single-atom calculations of H, using PWSP pseudopotentials, resulted in errors.
2. Optimized the HSE calculation code to avoid Intel errors.

### 10.2.6 2023A Updated on 2024/01/12

#### 10.2.6.1 IO adjustments

1. **Added Fermi level information to rho.bin.**
    a. When task=dos/band, the Fermi level can be directly read from rho.bin without reading from system.json; (This version is compatible with older versions of rho.bin that do not contain EFermi, but the older version of DS-PAW cannot read the rho.bin output by this version)
    b. Added parameter band.EfShift, which controls whether to read EFermi from rho.bin when task=band is used.
2. Supplement the ##PARAMETERS## section of DS-PAW.log with io.band and io.dos output.

3. Added band information output to DS-PAW.log

#### 10.2.6.2 Feature optimization

1. Fixed an issue where the partial results were incorrect when task=pcharge;
2. Added parameter scf.timeStep to adjust the convergence of electronic steps when cal.methods is set to 4 or 5.
3. **Added parameter *task=optical*.**
    a. Renamed the parameter cal.opticalGrid to optical.grid
    b. Added parameters optical.KKEta, optical.smearing, optical.sigma, optical.Emax

### 10.2.7 2023A Updated on 2023/10/07

#### 10.2.7.1 IO adjustments

1. Fixed an issue where a warning would be incorrectly output when *sys.functional* was not defined in the input file when *sys.hybrid* = *true*.
2. Fixed an issue where the number of FinalStep outputs did not match the number of step-XX outputs when task=aimd/relax.

#### 10.2.7.2 Feature optimization

1. Optimized the acoustic calculation module to strictly adhere to the acoustic sum rule.
2. Fix the issue of abnormal forces in the complex density functional task=relax.

### 10.2.8 2023A Updated on 2023/6/21

#### 10.2.8.1 IO Adjustment

1. Add force-related outputs in neb0X.h5 during NEB calculations.
2. Change E2095 error to warning and add relevant explanations
3. Adjust the output format for task=wannier
4. Increase the number of significant digits for reading and writing sys.fixedP-related data and parameter passing.

#### 10.2.8.2 Feature Optimization

1. Optimized the band.unfolding algorithm to reduce the probability of memory crashes.
2. In the *FixedPotential* iteration, precision control is implemented for the output *.input.json* file to prevent redundant calculations in some cases.
3. Fixed data format issues in some .txt output files, preventing data output stacking.

### 10.2.9 2023A Update: May 9, 2023

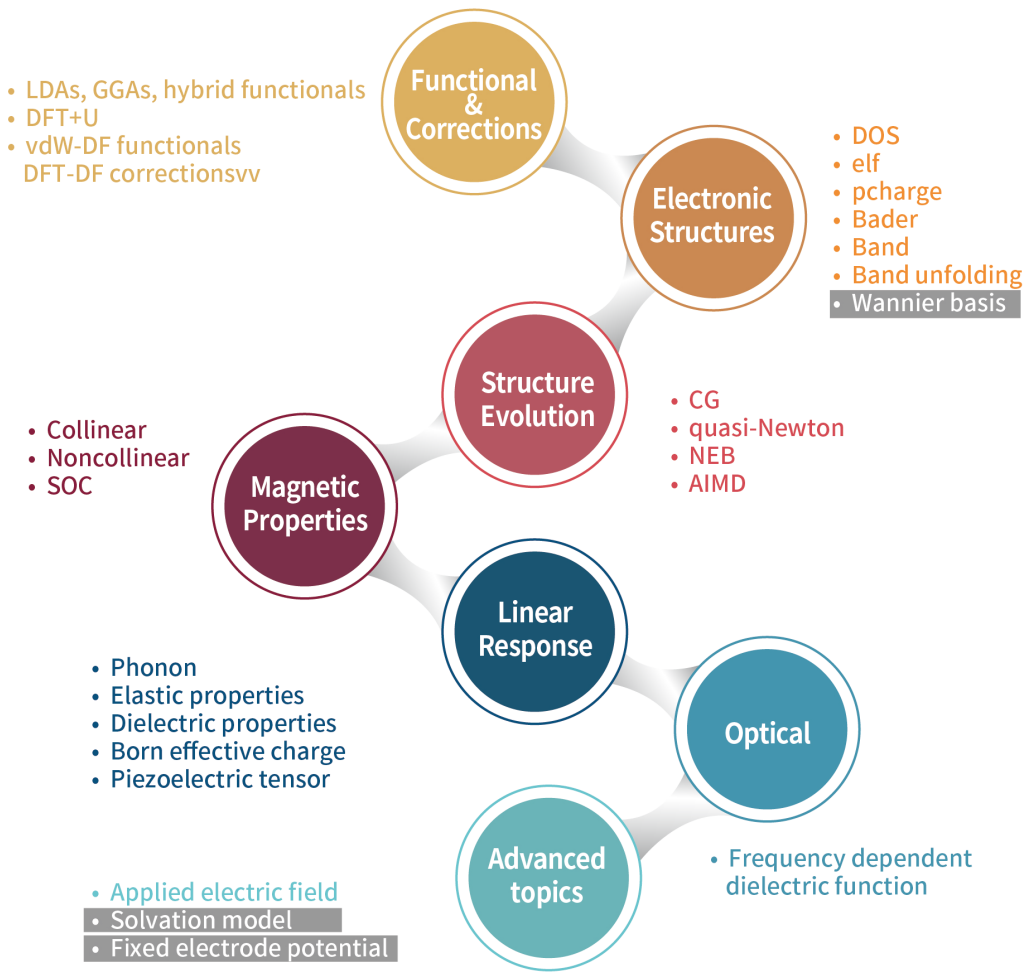
#### 10.2.9.1 I/O Adjustments

1. Adjusted output information related to task=neb, sys.hybrid=true, and scf.mixType=Broyden.
2. Fixed the error where HeatCapacity output was null when task=phonon and phonon.thermal=true.

#### 10.2.9.2 Feature Enhancements

1. Added fixed lattice and atom position information to latestStructure.as; fixed a bug related to incorrect mag information output.
2. Adjusted the io.magProject default value to true when sys.spin is set to collinear/non-collinear.
3. Fixed an issue with incorrect reading of mag-related information from relax.json/relax.h5 during continued calculations.

## 10.3 Function Summary




The functions highlighted in gray will be supported from DS-PAW 2023A and later versions.

## 10.4 Release History

### 10.4.1 2022A

#### 10.4.1.1 New Features

1. Support for revPBE/PBEsol/RPBE exchange-correlation functionals
2. Supports vdW functionals: vdW-optPBE, vdW-optB88, vdW-optB86b, vdW-DF, vdW-DF2, and vdW-revDF2
3. Supports simulation of external electric field effects
4. Supports NEB calculations with variable lattice systems (solid state NEB, ssNEB)
5. Supports calculation of ferroelectric polarization using modern polarization theory
6. Support band unfolding functionality
7. Support calculation of Helmholtz free energy/constant volume heat capacity/entropy using force constant matrix
8. Support calculation of phonon band with long-range Coulomb interaction considered
9. Support calculation of dielectric tensors using the linear response method
10. Support calculation of piezoelectric tensors using the linear response method
11. Support calculation of Born effective charges using the linear response method
12. Support Bader charge analysis
13. Support constraining lattice degrees of freedom along specified dimensions during structure optimization.

---

#### 10.4.1.2 Feature Optimization

1. Supported .paw (Hongzhiwei PAW pseudopotential format) / .potcar (VASP POTCAR pseudopotential format) / .pawpsp (GBRV PAW pseudopotential format)
2. Added a preconditioned conjugate gradient method in the self-consistent iteration algorithm.
3. Added a fast inertial relaxation method in NEB relaxation.
4. Added a convergence criterion option for energy convergence in structure relaxation and NEB calculations.
5. Added support for modifying the Alpha and Omega coefficients in hybrid functionals, and accelerated hybrid functional calculations using the Adaptively Compressed Exchange Operator.
6. Added projected magnetic moment information, maximum force during structure relaxation, maximum force during transition state search, and band gap information in the output file.
7. Added a temporary calculation folder paw_tmp within the calculation directory to store intermediate files and error messages.

---

### 10.4.2 2021B

#### 10.4.2.1 New Features

1. Support for CI-NEB method for transition state search
2. Supports hybrid functionals PBE0, HSE03, and HSE06.
3. Support DFT-D2 and DFT-D3 van der Waals corrections

4. Supports calculation of dielectric constant, refractive index, reflectivity, absorption coefficient, extinction coefficient, and more
5. Support calculations for charged systems.
6. Support spin-orbit coupling
7. Support phonon band structure and density of states calculations using the finite displacement method.
8. Support phonon band structure and density of states calculations using the DFPT method
9. Support DFT+U for strongly correlated systems
10. Support first-principles molecular dynamics calculations

---

### 10.4.3 2021beta

#### 10.4.3.1 New Features

1. Using a plane-wave basis set to expand the wavefunctions
2. Using the Projector Augmented-Wave (PAW) method for pseudopotentials
3. Structure relaxation calculations, supporting atomic position relaxation, lattice relaxation, and lattice and atomic position relaxation
4. Self-consistent field (SCF) calculations
5. Support non-spin-polarized, collinear spin-polarized, non-collinear spin-polarized, and spin-orbit coupling systems
6. Total energy calculation
7. Atomic force calculation
8. Stress calculation
9. Band structure (projected band structure) calculation
10. Electronic density of states (projected density of states) calculation
11. Electron Localization Function (ELF) calculation
12. Potential calculation, supporting electrostatic potential and local potential calculations