T4ME Documentation

Release 2.0.0

Espen Flage-Larsen

Nov 21, 2019

Contents

1 Features 3

2 Structure 5

3 Contributing and versioning 7

4 Author 9

5 License 11

6 Documentation 13

Python Module Index 121

Index 123

```
  ______________________   _____   ____       ___________________
 /                   /   / /   / /   ___/   /              /
/_____      _____/   / /   / /          /   _________/
     /   /   /   /_/   /_/          /   /____
    /   /   /           /   /_/   /   _____/
   /   /   /______     ___/   /   /   /   /_________
  /   /          /   / /   /   /   /             /
/____/          /____/ /____/  /____/______________/
```

Routines to calculate the transport properties of materials using the linearized Boltzmann Transport Equations (BTE) in the Relaxtion-Time-Approximation (RTA).

Please go to the T4ME documentation for more extensive documentation and information regarding usage (the API documentation is currently not operational).

CHAPTER 1

Features

- Modular, easily extendable by users
- Band structures:
  - Generate the band structure from analytic function
    - Parabolic bands
    - Parabolic bands pluss a quartic correction
    - Kane type of bands
  - Read from first-principle codes
    - Interface to VASP is included
    - Interface to read Wannier90 input and output files and use these to construct tight binding orbitals using PythTB is included.
  - Read from NumPy datafiles
- Scattering properties:
  - Parabolic energy dispersion models:
    - Acoustic phonon scattering from deformations
    - Non-polar optical phonon scattering (not fully tested)
    - Piezoelectric acoustic phonon scattering (not fully tested)
    - Polar optical phonon scattering (not fully tested)
    - Intervalley phonon scattering (not fully tested)
    - Ionized impurity scattering
  - Density of states models: - Acoustic phonon scattering from deformations - Non-polar optical phonon scattering (not fully tested) - Polar optical phonon scattering (not fully tested) - Intervalley phonon scattering (not fully tested)
  - Alloy scattering

- Solution of the transport and density of states integrals:
  - Trapezoidal, Simpson and Romberg integration of a static input grid
  - Linear tetrahedron method (Spglib needed)
  - Weighed sum method
- Interpolation of the band structure and scattering properties:
  - All routines available in SciPy
  - GeometricTools/WildMagic regular grid routines

CHAPTER 2

Structure

The structure of the program is simple: the main routines are written in Python utlizing NumPy and SciPy where necessary. In addition there are calls to external routines through Cython, particularly the optional libraries. Only support for Python3 is confirmed.

CHAPTER 3

Contributing and versioning

Standard Git versioning is utilized. Contributions are welcome, encouraged and (greatly) appreciated. Please go here: T4ME@GitHub

CHAPTER 4

Author

Espen Flage-Larsen with funding from the Norwegian Research Council, Thelma project (228854).

CHAPTER 5

License

This project is licensed under the BSD 3-clause license. Please see `LICENSE.md` included in the root folder of T4ME for additional details.

CHAPTER 6

Documentation

6.1 Prerequisites

In its basic form T4ME only need the following dependency:

- Spglib. This can be installed with

```
pip install spglib
```

Additional optional dependencies include:

- Spglib, A custom interface to Spglib to enable tetrahedron integration.
- GeometricTools (use Wildmagic 5.14). Used to interpolate band structure data, for instance by offering Akima interpolation.

6.2 Download

If using *pip* it is not necessary to download the package, it can simply be installed using

```
pip install T4ME
```

Otherwise, T4ME is hosted at GitHub and can be obtain by

```
git clone git@github.com:espenfl/t4me.git
```

or

```
git clone https://github.com/espenfl/t4me.git
```

6.3 Installing

6.3.1 Basic install

First make sure Spglib is installed

```
pip install spglib
```

Then install T4ME by executing the command

```
pip install T4ME
```

This will give the user the posibility to calculate the transport coefficients using integration routines in SciPy. For other integration and interpolation routines the user needs to follow the following recipe.

6.3.2 Advanced install

For more advanced functionality (interpolation and other integration routines) the user should determine which external libraries are needed and install them based on their respective documentation. Please also fetch the repository from github and work from its base directory when executing the following commands.

The `setup.py` file assume in its supplied form that the user installs the libraries in the standard folders, e.g. *$HOME/include* and *$HOME/lib* for the include and library files, respectively. If other locations are needed, please adapt the `setup.py` file.

As an example, we want to enable the tetrahedron integration. A Spglib interface needs to be compiled. This can be build with the included `build_spglib` file.

```
./build_spglib
```

If that was successfull, T4ME can then be built by issuing the following command

```
pip install .
```

or

```
pip install -e .[dev]
```

Another example. We want to enable SKW interpolation. The SKW routines can be built (assuming Intel MKL is installed) by issuing

```
./build_skw
```

If successfull, T4ME can then be installed by issuing one of the two commands listed above. If other FFT routines are to be used, please modify `skw/Makefile`.

All other libraries need to be built externally and linked in.

Upon successfull completion of the installation, T4ME is executed with the command

```
t4me
```

An *input* directory is needed which should contain the input files.

6.4 Running tests

Several tests are included, which can be used to test the installation.

Currently only basic functionality is tested and developers are encouraged to write tests for any added functionality.

Tests are executed by issuing

```
pytest
```

In the base directory.

6.5 Input

The program relies on three parameter files written in YAML. They should all reside inside the *input* directory.

- `input/param.yml` - contains the main parameters, which routines to execute, what type of integration, interpolation etc. to perform and what input files to use.
- `input/bandparam.yml` - contains parameters for band generation, the scattering parameters for each band and the parameters to use in the tight binding generation (if that is needed).
- `input/cellparam.yml` - contains details of the unit cell, its atoms and the reciprocal sampling density.

In addition if external input files be used they should also be placed in the *input* folder.

6.5.1 The input parameters

General input parameters

- *Notes about format*
- *Dispersion relations*
  - *dispersion_interpolate*
  - *dispersion_interpolate_sampling*
  - *dispersion_interpolate_step_size*
  - *dispersion_interpolate_method*
  - *dispersion_interpolate_type*
  - *dispersion_velocities_numdiff*
  - *dispersion_write_preinter*
  - *dispersion_write_postinter*
  - *dispersion_write_start*
  - *dispersion_write_end*
  - *dispersion_num_kpoints_along_line*
  - *dispersion_effmass*
  - *dispersion_effmass_diagonalize*

  - *dispersion_effmass_transform*
- *Electron transport*
  - *transport_calc*
  - *transport_method*
  - *transport_integration_method*
  - *transport_integration_spectral_smearing*
  - *transport_integration_spectral_density*
  - *transport_integration_spectral_energy_cutoff*
  - *transport_chempot_min*
  - *transport_chempot_max*
  - *transport_chempot_samples*
  - *transport_energycutband*
  - *transport_include_bands*
  - *transport_use_analytic_scattering*
  - *transport_drop_valence*
  - *transport_drop_conduction*
- *Density of states*
  - *dos_calc*
  - *dos_e_min*
  - *dos_e_max*
  - *dos_num_samples*
  - *dos_smearing*
  - *dos_integrating_method*
- *General parameters*
  - *temperature_min*
  - *temperature_max*
  - *temperature_steps*
  - *gamma_center*
  - *maxeint*
  - *occ_cutoff*
  - *e_fermi_in_gap*
  - *e_fermi*
  - *e_vbm*
  - *e_shift*
  - *skw_expansion_factor*

- *carrier_valence_energy*
- *carrier_conduction_energy*
- *carrier_dos_analytick*
- *defect_ionization*
- *donor_number*
- *donor_degen_fact*
- *donor_energy*
- *acceptor_number*
- *acceptor_degen_fact*
- *acceptor_energy*
- *read*
- *readfile*
- *scissor*
- *symprec*
- *libinfo*
- *onlytotalrate*
- *parallel*
- *run_tests*

Notes about format

The input files follow normal YAML conventions. Please inspect the sample file `input/param.yml`. Even though many parameters have default values if not specified the user should always run the calculations with fully specified input files for consistency and reproducibility.

Dispersion relations

The following parameters are related to the energy and velocity dispersion relations.

`dispersion_interpolate`

If set to `True` the band structure is interpolated on a k-point grid.

Example:

```
dispersion_interpolate: False
```

Do not interpolated the band structure.

`dispersion_interpolate_sampling`

The target k-point sampling when performing interpolation.

Example:

```
dispersion_interpolate_sampling: [45,45,45]
```

Interpolates the input band structure to a grid density of 45, 45 and 45 k-points along the unit axis of the supplied k-point grid.

`dispersion_interpolate_step_size`

The target k-point step size in inverse AA. In order for this parameter to work, the user have to set

```
dispersion_interpolate_sampling: [0,0,0]
```

Example:

```
dispersion_interpolate_sampling: [0.1,0.1,0.1]
```

Creates a k-point sampling that is at least as dense as to give a step size of 0.1 inverse AA between each k-point along each reciprocal axis.

`dispersion_interpolate_method`

Choses which interpolative method to use. The following options are currently available:

- *linearnd* - Uses `LinearNDInterpolator` in SciPy.
- *interpn* - Uses `interpn` in Scipy.
- *rbf* - Uses the Scipy version, but that is memory intensive
- *wildmagic* - Uses the GeometricTools (former WildMagic) interpolation routines.
- *skw* - Uses Fourier interpolation
- *tb* - Extracts the energies on a denser grid from a tight- binding model

Tests have shown that the last three methods are quite general and, given what they are, quite accurate.

Example:

```
dispersion_interpolate_method: "wildmagic"
```

Will for instance use the WildMagic library.

`dispersion_interpolate_type`

Additional selective layer for the method chosen by :ref'*dispersion_interpolate_method*. Currently, the following options are availble:

- *nearest* or *linear* - if `dispersion_interpolate_method` = *linearnd*
- *trilinear*, *tricubic_exact*, *tricubic_bspline*, *akima* - if `dispersion_interpolate_method` = *wildmagic*

Example:

```
dispersion_interpolate_type: "akima"
```

Uses the Akima interpolation in the WildMagic library.

`dispersion_velocities_numdiff`

Use numerical differentiation to calculate the velocities if they are not present on entry, or/and use numerical differentiation to extract the velocities after the dispersions have been interpolated (used by default for the interpolat routines that do not support velocity extraction)

Example:

```
dispersion_velocities_numdiff: False
```

Turns for instance of the numerical difference calculation of the velocities. In this case please make sure that the velocities are present on input or that they are genrated by other means.

`dispersion_write_preinter`

Selects if a line extraction of the band structure is written to the file `bands` before interpolation. If velocities are present this is also written to the file `velocities`

Example:

```
dispersion_write_preinter: False
```

Writes the extracted band structure values along a line to file(s).

`dispersion_write_postinter`

Selects if a line extraction of the band structure is written to the file `bands_inter` after interpolation. If velocities are present this is also written to the file `velocities_inter`

Example:

```
dispersion_write_postinter: False
```

Does not write the extracted band structure values along a line to file(s).

`dispersion_write_start`

The start point (in direct coordinates) for the line extraction.

Example:

```
dispersion_write_start: [0.0, 0.0, 0.0]
```

An example start point, here the Gamma point.

dispersion_write_end

The end point (in direct coordinates) for the line extraction.

Example:

```
dispersion_write_end: [0.5, 0.0, 0.0]
```

dispersion_num_kpoints_along_line

How many samples to use along the line to be extracted.

Example:

```
dispersion_num_kpoints_along_line: 20
```

Here 20 points is used along the line.

dispersion_effmass

Calculate the effective mass tensor along the unit vectors of the configured reciprocal cell. The resulting tensor is in units of the free electron mass. Currently it is not printed out and an error will occur.

Example:

```
dispersion_effmass: False
```

Do not calculate the effective mass tensor.

dispersion_effmass_diagonalize

Diagonalize the calculated effective mass tensor. Currently the diagonal elements and the eigenvectors are not printed out and an error will occur.

Example:

```
dispersion_effmass_diagonalize: False
```

Do not diagonalize the effective mass tensor.

dispersion_effmass_transform

The transformation vectors for the effective mass tensor. The elements [0,:] give the first vector, [1,:] the second and [2,:] the third. Should be given in direct coordinates. If the array is left empty, no transformation is performed.

Example:

```
dispersion_effmass_transform: []
```

Do not transform the effective mass tensor.

Electron transport

The following parameters determines how the transport of electrons is to be determined.

`transport_calc`

Determines if the transport calculations are to executed.

Example:

```
transport_calc: True
```

Calculate the transport properties.

`transport_method`

Selects which mode to use to calculate the transport properties. Currently three different modes are accepted;

- *closed* - The integrals are solved using the closed Fermi-Dirac integrals. Only available if the band structure is generated by means of analytic models. Only one scattering mechnism can be used for each band in this approach.
- *numeric* - A numerical integration of the Fermi-Dirac integrals, which allows to concatenate different scattering mechanisms for each band.
- *numerick* - The integrals are solved by integrating over the k-point grid or by utilizing the spectral function.

Example:

```
transport_method: "numerick"
```

In this example the transport integrals are solved using the closed analytical expressions for the Fermi-Dirac integrals.

`transport_integration_method`

Selects which method to use for solving the integral over the k-points. Only applicable if `transport_method` is set to *numerick*.

- *trapz* - Use the trapezoidal integration scheme implemented in SciPy
- *simps* - Use the Simpson integration scheme implemented in SciPy
- *romberg* - Use the Romberg integration scheme implemented in SciPy
- *tetra* - Use the linear tetrahedron method
- *smeared* - Use the weighted sum approach with a smearing factor

`transport_integration_spectral_smearing`

Gaussian smearing factor for the weighted sum approach. In units of eV. Only relevant if `transport_integration_method` is set to *smeared*.

Example:

```
transport_integration_spectral_smearing: 0.1
```

Would set it to 0.1 eV.

`transport_integration_spectral_density`

The sampling density of the spectral function. Only relevant if `transport_integration_method` is set to *tetra* or *smeared*.

Example:

```
transport_integration_spectral_density: 1000
```

An example requesting 1000 samples.

`transport_integration_spectral_energy_cutoff`

Determines the extra padding that is used for the spectral function on both sides of the requested chemical potential. If multiple chemical potentials are requested, the lowest and the highest value is checked and the range of the energy interval on which the spectral function is calculated is padded with the specified value. Only relevant if `transport_integration_method` is set to *tetra* or *smeared*. In units of eV.

Example:

```
transport_integration_spectral_energy_cutoff: 1.0
```

Here, 1.0 eV is subtracted (added) to the smallest (largest) requested chemical potential.

`transport_chempot_min`

The minimum chemical potential requested for which the transport coefficients are calculated. In units of eV.

Example:

```
transport_chempot_min: -1.0
```

Starts the calculation of the transport properties at -1.0 eV.

`transport_chempot_max`

The maximum chemical potential requested for which the transport coefficients are calculated. In units of eV.

Example:

```
transport_chempot_max: 1.0
```

Ends the calculation of the transport properties at 1.0 eV.

`transport_chempot_samples`

The number of chemical potential samples to use between `transport_chempot_min` and `transport_chempot_max`.

Example:

```
transport_chempot_samples: 100
```

Extract the transport coefficients at 100 points between `transport_chempot_min` and `transport_chempot_max`.

`transport_energycutband`

Bands that reside `transport_energycutband` outside the chemical potential is dropped from the calculation of the transport coefficients. All k-points are currently analyzed in order to determine which bands fall inside the energy range [`transport_chempot_min-transport_energycutband`,``transport_chempot_max``+``transport_energycutband``] . Units in eV.

Example:

```
transport_energycutband: 1.0
```

Substract and add 1.0 eV to `transport_chempot_min` and `transport_chempot_max`, respectively. Bands that does not have any k-point with energy in the range [-2.0 eV, 2.0 eV] is not included in the calculation of the transport coefficients.

`transport_include_bands`

A list containing specific bands on which to calculate the transport coefficients. If the list is empty, use all bands within the range set by :ref:`transport_energycutband`. Band index starts at 1.

Example:

```
transport_include_bands: [3, 4, 10]
```

Calculate the transport coefficients for band 3, 4 and 10.

`transport_use_analytic_scattering`

Determines if the analytic parabolic scattering models should be used. They can be applied also to dispersions which are not parabolic, but such an application have to be physically justified.

Example:

```
transport_use_analytic_scattering: False
```

Use the density-of-states to set up the scattering mechanisms.

`transport_drop_valence`

Determines if all valence band should be dropped while reading e.g. external data. Currently only works for the VASP interface.

Example:

```
transport_drop_valence: False
```

Do not exclude the valence bands during read-in.

`transport_drop_conduction`

Determines if all conduction bands should be dropped while reading e.g. external data. Currently only works for the VASP interface.

Example:

```
transport_drop_conduction: False
```

Do not exclude the conduction bands during read-in.

Density of states

Here follows input parameters related to the calculation of the density of states.

`dos_calc`

Determines if the user wants to calculate the density of states. Even if this flag is set to *False*, the density of states is sometimes calculated if needed, e.g. if the density of states dependent scattering models are employed. However, with this parameter set to *True* and e.g. `transport_calc` set to *False* it is possible to only calculate the density of states.

```
dos_calc: False
```

Do not calculate the density of states.

`dos_e_min`

The minimum energy to use for the density of states calculation. In units of eV. The reference is with respect to the aligned Fermi level and consequetive shift that might have been applied. Note that the range of density of states calculation might change if it is called from other routines, e.g. the density of states dependent scattering models in order to cover enough energies.

```
dos_e_min: -5.0
```

Calculate the density of states from -5.0 eV.

`dos_e_max`

The maximum energy to use for the density of states calculation. In units of eV. The reference is with respect to the aligned Fermi level and consequetive shift that might have been applied. Note that the range of density of states calculation might change if it is called from other routines, e.g. the density of states dependent scattering models in order to cover enough energies.

```
dos_e_max: 2.0
```

Calculate the density of states to 2.0 eV.

dos_num_samples

The number of energy samples between `dos_e_min` and `dos_e_max`.

```
dos_num_samples: 1000
```

Use 1000 energy points from `dos_e_min` to `dos_e_max`.

dos_smearing

Gaussian smearing factor in units of eV. Only relevant if `dos_integrating_method` is set to *smeared*, *trapz*, *simps* or *romb*.

```
dos_smearing: 0.1
```

dos_integrating_method

Determines which method of integration to use to obtain the density of states. The following options are available:

- *trapz* - trapezoidal integration
- *simps* - Simpson integration
- *romb* - Romberg integration
- *tetra* - linear tetrahedron method without Blochl corrections

```
dos_integrating_method: "trapz"
```

Use trapezoidal integration to obtain the density of states.

General parameters

Here follows general parameters.

temperature_min

The minimum temperature in K.

Example:

```
temperature_min: 100
```

The minimum temperature is set at 100 K.

`temperature_max`

The maximum temperature in K.

Example:

```
temperature_max: 700
```

The maximum temperature is set at 700 K.

`temperature_steps`

The number of temperature steps from `temperature_min` to `temperature_max`.

Example:

```
temperature_steps: 7
```

In total 7 temperature steps, resulting in temperature samplings at 100, 200, 300, 400, 500, 600 and 700 K.

`gamma_center`

$Gamma$ centered k-point grids? Anything else is currently not supported (or tested).

Example:

```
gamma_center: True
```

Notifies that the k-point grids are
$Gamma$ centered.

`maxeint`

The limites of the dimensionless carrier energy
eta used for the numerical solution of the Fermi-Dirac integrals. Only relevant if `transport_method` is set to *numerick*.

Example:

```
maxeint: 100
```

Sets the limits of the Fermi-Dirac integrals to 100
eta.

`occ_cutoff`

The cutoff to use when detecting occupancies. Used for detecting the valence band maximum, conduction band minimum and then also for the band gap.

Example:

```
occ_cutoff: 1.0e-4
```

The occupancy cutoff is set at 1.0e-4, which means that states with an occupancy less than this will be assumed not occupied and vice versa.

e_fermi_in_gap

Determines if the Fermi level is to be placed in the middle of the gap.

Example:

```
e_fermi_in_gap: False
```

Do not place the Fermi level in the middle of the gap.

e_fermi

Determine if one should shift the energies to the supplied Fermi level (usually read in the interface).

Example:

```
e_fermi: True
```

Shift the energies such that zero is placed at the supplied Fermi level.

e_vbm

Determines if to set the Fermi level at the valence band maximum.

Example:

```
e_vbm: False
```

Do not set the Fermi level at the top valence band.

e_shift

After all alignments have been performed, perform this additional shift. Units in eV.

Example:

```
e_shift: 0.0
```

Sets the additional energy shift to 0 eV.

skw_expansion_factor

The expansion factor used in the SKW routine. It is basically tells how many unit cells that can be used. Only relevant if `dispersion_interpolate_method` is set to *skw*.

Example:

```
skw_expansion_factor: 5
```

Use 5 unit cells in each direction. In a second step a sphere is cut from this volume, thus removing the points in the far corners of this volume in the interpolation procedure.

carrier_valence_energy

The cutoff in which where to interpret the carriers as p-type. Used in the calculation of the carrier concentration. Units in eV.

Example:

```
carrier_valence_energy: 0.0
```

Would make sure all carriers at negative energies are interpreted as p-type.

carrier_conduction_energy

The cutoff in which where to interpret the carriers as n-type. Used in the calculation of the carrier concentration. Units in eV.

Example:

```
carrier_valence_energy: 0.0
```

Would make sure all carriers at positive energies are interpreted as n-type.

carrier_dos_analytick

Determines if the carrier concentration should be recaculated after being set up with analytical models. Only relevant if the band structure is generated from analytical models.

Example:

```
carrier_dos_analytick: True
```

Do not recalculate and use the analytical expressions for the carrier concentration.

defect_ionization

Determines if we shoudl use the expressions for the defect ionization in order to calculate the p- and n-type carrier concentration.

Example:

```
defect_ionization: False
```

Do not use the models for the defect_ionization to adjust the p- and n-type carrier concentration.

donor_number

The density of donors in units of $10^{-21}\mathrm{cm}^{-3}$.

Example:

```
donor_number: 0.0
```

No donors present.

donor_degen_fact

The degeneracy factor for the donors.

Example:

```
donor_degen_fact: 0.75
```

A degeneracy factor of 0.75 is used.

donor_energy

The energy of the donor in units of eV. Should be referenced to the energy after all adjustments to the Fermi level and additional energy shifts have been performed.

Example:

```
donor_energy: 0.0
```

The donor energy is 0 eV.

acceptor_number

The density of acceptors in units of $10^{-21}\mathrm{cm}^{-3}$.

Example:

```
donor_number: 0.0
```

No acceptors present.

acceptor_degen_fact

The degeneracy factor for the acceptors.

Example:

```
acceptor_degen_fact: 0.75
```

A degeneracy factor of 0.75 is used.

acceptor_energy

The energy of the acceptor in units of eV. Should be referenced to the energy after all adjustments to the Fermi level and additional energy shifts have been performed.

Example:

```
acceptor_energy: 0.0
```

The acceptor energy is 0 eV.

read

Determine how to set up the band structure and/or how to read data. The following options are possible:

- *param* - The band structure is generated from the parameter files. For all cases the band structure is generated by analytical models. The parameters pertaining to the construction of the bandstructure itself is set in the file `bandparam.yml`.
- *numpy* - Read data from NumPy datafiles without group velocities.

  The datastructure of the supplied numpy array
  should be on the following format:
  [
  [kx], [ky], [kz], [e_1], [v_x_1], [v_y_1], [v_z_1],
  [e_2], [v_x_2], [v_y_2], [v_z_2], … ,
  [e_n], [v_x_n], [v_y_n], [v_z_n]
  ]

  The band parameters still need to be set in `bandparam.yml` as they contain necessary information about scattering etc.
- *numpyv* - Read data from NumPy datafiles, including group velocities.

  The datastructure of the supplied numpy array
  should be on the following format:
  [
  [kx], [ky], [kz], [e_1], [e_2], … , [e_n]
  ]

  The band parameters still need to be set in `bandparam.yml` as they contain necessary information about scattering etc.
- *vasp* - Read data from a supplied VASP XML file, typically vasprun.xml. The band parameters still need to be set in `bandparam.yml` as they contain necessary information about scattering etc.

Example:

```
read: param
```

Construct the band structure from the parameters present in `bandparam.yml`.

`readfile`

The name of the file to be read. Depending on `read` it has the following behaviour:

- *param* - not relevant
- *vasp* - the name of the VASP XML file, if not set it defaults to *vasprun.xml*
- *numpy* - the name of the NumPy datafile
- *numpyv* - the name of the NumPy datafile

Example:

```
readfile: ""
```

Use defaults, e.g. vasprun.xml for VASP.

`scissor`

Apply a simple scissor operator to increase the band gap. Only works of the band gap has been correctly determined. In units of eV if not *False*.

Example:

```
scissor: False
```

Do not apply a scissor operator.

`symprec`

The symmetry cutoff parameters. Passed to Spglib. VASP also uses an internal symmetry parameter which is called *SYMPREC*. Spglib need to reproduce the symmetry that was detected in VASP in order for the k-point grids and thus the mapping between the IBZ and BZ to be valid. If errors regarding this is invoked, please try to adjust symprec.

Example:

```
symprec: 1.0e-6
```

If two coordinates are within 1.0e-6 it is assumed that they are the same and symmetry is thus detected.

`libinfo`

Determines if printout to stdout is performed in the interfaces to the external libraries.

Example:

```
libinfo: False
```

Do not print stdout information from the interfaces.

`onlytotalrate`

Determines if the users wants to store the relaxation time for each scattering mechanism. This is usefull for visualization purposes, but is simply very memory demanding. Users should try to leave this to *True*.

Example:

```
onlytotalrate: True
```

Only store the total relaxation time.

`parallel`

Determines if transport and density of states integrals are to be performed in parallel (embarrassingly). Currently this is not fully implemented, so users should leave this to *False*.

Example:

```
parallel: False
```

Do not use the parallel features.

`run_tests`

Determines if the tests are to be run. Several options are available:

- *slow* - Run all tests.
- *fast* - Only run the fast tests.
- *True* - Same as *fast*.
- *False* - Do not run any tests.

Example:

```
run_tests: False
```

Do not run any tests.

Band structure input parameters

- *Notes about format*
- *General parameters*
  - *`type`*
  - *`effmass`*
  - *`a`*
  - *`e0`*
  - *`status`*

  - `kshift`
  - `spin_degen`
- *General scattering related parameters*
  - `select_scattering`
  - `explicit_prefact`
  - `explicit_prefact_values`
- *Acoustic phonon scattering parameters*
  - `d_a`
  - `speed_sound`
- *Piezoelectric phonon scattering parameters*
  - `p`
  - `isl`
- *Non-polar optical phonon scattering*
  - `d_o`
  - `n_o`
  - `omega_o`
- *Polar optical phonon scattering*
  - `epsi`
  - `f`
- *Intervalley acoustic phonon scattering*
  - `n_vv`
  - `omega_vv`
  - `etrans`
  - `zf`
  - `q_energy_trans`
- *Ionized impurity scattering parameters*
  - `n_i`
  - `isl_i`
  - `z`
- *Alloy scattering parameters*
  - `vdiff`
  - `alloyconc`
- *Common scattering parameters*
  - `eps`
  - `rho`

- *tau0_c*
- *emmission*

Notes about format

The input files follow normal YAML conventions. Please inspect the sample file `input/bandparam.yml`. Even though many parameters have default values if not specified the user should always run the calculations with fully specified input files for consistency and reproducibility.

There is one entry per band. If many bands are used one can specify a range, e.g. Band X-Y: to set the same parameters for bands X to Y. Or if one would want to set the same parameters for all bands one should use Band 1-: This is quite usefull when reading data from a full-band calculation of some sort.

Remember to use two spaces indent after each Band entry (before a new Band entry) in order to comply with the YAML formatting standard.

Also, the parameters should be indented with two spaces from the Band entry:

Band 1-2:
  aparameter: somevalue
  anotherparameter: someothervalue

Band 3-5:
  aparameter: somevalue
  anotherparameter: someothervalue

Make sure all bands are specified. In this example, five bands was included.

General parameters

The following parameters are general and does not relate directly to a specific scattering mechanism etc.

`type`

Determines how to generate the bands if not read. Relevant only if `read` is set to *param*. The following options are possible:

- *0* - parabolic bands according to the relation

  where the effective mass m is set by `effmass`.

- *1* - parabolic bands pluss a quartic correction according to the relation:

  where the effective mass m is set by `effmass` and the correction factor a is set by `a`.

- *2* - Kane types (alpha correction) according to the relation:

  where the effective mass m is set by `effmass` and the correction factor $alpha$ is set by `a`.

No band folding is performed, except for the tight-binding case. It is important thus to scale the unit cell such that there is enough band coverage within the requested region of the chemical potentials pluss the excess needed for the thermal broadening.

`effmass`

The effective mass in units of the free electron mass along each configured unit vector in reciprocal cell. Use negative values to generate bands that curve down and vice versa.

Example:

```
effmass: [-1.0,-1.0,-1.0]
```

Generates band that for the parabolic case curves down with an effective mass along each unit vector of the configured recirprocal cell equal to the free electron mass.

`a`

The correction factor to be applied. See `type` for additional description. Is given along each unit vector in the configured reciprocal cell similar to the effective mass.

Example:

```
a: [-100.0,-100.0,-100.0]
```

Applies a correction factor of -100.0 along each unit vector direction in the currently configured reciprocal cell.

`e0`

An energy shift in units of eV. Applies to the current band.

Example:

```
e0: 0.0
```

Shift the band with 0.0 eV.

`status`

Determines if this is a valence or a conduction band. The following options are available:

- *v* - valence band
- *c* - conduction band

Example:

```
status: v
```

This band is a valence band.

kshift

Shift the band by a reciprocal vector, otherwise it is centered at Gamma. Have to be specified in cartesian coordinates.

Example:

```
kshift: [0.0,0.0,0.0]
```

Do not apply any shift to the current band.

spin_degen

The spin degeneracy of the current band. The following options are available:

- *1* - not spin degenerated
- *2* - spin degenerated

Example:

```
spin_degen: 2
```

The current band is spin degenerated.

General scattering related parameters

In the following the parameters related to the setup of the scattering mechanisms are given.

select_scattering

Determines which scattering mechnisms to apply for the current band. Set element to 1 to include scattering, 0 otherwise. Currently the following scattering mechanisms have been implemented (the number indicate array index, starting at 1):

- 1 elastic acoustic phonon scattering from def. pot.
- 2 non-polar optical phonon scattering
- 3 intervalley phonon scattering
- 4 polar optical phonon scattering
- 5 piezoelectric phonon scattering
- 6 ionized impurity scattering (Brooks-Herring)
- 7 ionized impority scattering (Conwell-Weiskopf)
- 8 alloy scattering
- 9-11 empty slots
- 12 constant scattering

If one does not use the analytic (parabolic) scattering models and instead use the density of states to generate the scattering rate, then only the first four and the last have been implemented (currently only the first and last have been properly tested)

Example:

```
select_scattering: [1,0,0,0,0,0,0,0,0,0,0,0]
```

Apply acoustic-phonon scattering by deformation potential to the following band.

explicit_prefact

Set an explicit prefactor for the relaxation time instead of using the prefactor from the density of states or parabolic band models. This behavior is enabled by setting the relevant element to *1* for the mechanism where one would like to specify an explicit prefact (constant tau0 is not included and is set below) for. Make sure that the total units that come out should be in fs. This is not always so easy to do due to temperature variations etc. Thus if the user also perform calculations at different temperatures, please consider that the prefactor usually change. This option should only be used by experts. If all elements in the array is *0*, the scattering models based on density of states or parabolic bands is used.

Example:

```
explicit_prefact: [0,0,0,0,0,0,0,0,0,0,0,0]
```

Disable the use of explicit prefactors.

explicit_prefact_values

The values of the explicit prefactors. Only relevant for the entries in `explicit_prefact` with a value of *1*. Remember that the units of the relaxation time come out as fs, including the energy dependency (density of states or parabolic band). Depending on the model, the prefactor thus have different units. Also consider that the prefactor usually has a temperature and effective mass dependence.

Example:

```
explicit_prefact_values: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                          0.0, 0.0, 0.0, 0.0, 0.0]
```

All explicit prefactors of the relaxtion time is set to zero.

Acoustic phonon scattering parameters

This model uses the linear Debye model.

d_a

Acoustic deformation potential in units of eV. Remember to rescale this is the overlap matrix is not one.

Example:

```
d_a: 10
```

Use a deformation potential of 10 eV.

`speed_sound`

The speed of sound. This is the group velocity of the low energy acoustic branch that is in the Debye model assumed to be linear. In units of m/s.

Example:

```
speed_sound: 10000
```

Use a group velocity of 10000 m/s.

Piezoelectric phonon scattering parameters

This model uses the polarization that is set up due to strain effects to describes acoustic phonon scattering. Typically important for polar materials.

`p`

The piezoelectric constant in units of C/m^2

Example:

```
p: 0.0
```

The piezoelectric constant is set to zero.

`isl`

The inverse screening length in the Debye formulation in units of inverse AA.

Example:

```
isl: 0.0
```

The inverse screening length is set to zero.

Non-polar optical phonon scattering

This model uses the Einstein model of a optical phonon mode (dispersion assumed to be flat so a constant value is used for the frequency).

`d_o`

The optical deformation potential in units of eV/AA.

Example:

```
d_o: 35.0
```

The optical deformation potential is set to 35.0 eV/AA.

`n_o`

The occupation number of the optical phonon.

Example:

```
n_o: 0.0
```

The occupation number of the optical phonon is set to zero.

`omega_o`

The optical phonon frequency to use from the Einstein model. In units of THz.

Example:

```
omega_o: 0.0
```

The optical phonon frequency is set to zero.

Polar optical phonon scattering

After the Froelich model. Should be replaced for a more explicit model in the future.

`epsi`

The permitivity of the electron in units of the vacuum permitivity.

Example:

```
epsi: 0.0
```

The permitivity is set to zero.

`f`

The Froehlich term.

Example:

```
f: 0.0
```

The Froechlich term is set to zero.

Intervalley acoustic phonon scattering

A model where the electron scatters both of acoustic and optical phonon modes. E.g. phonons connect two valleys.

n_vv

The intervalley phonon occupation number.

Example:

```
n_vv: 0.0
```

The intervalley phonon occupation number is set to zero.

omega_vv

The transition frequency in units of THz.

Example:

```
omega_vv: 0.0
```

The transition frequency is set to zero.

etrans

The transition energy between the bottom of the two values. In units of eV.

Example:

```
etrans: 0.0
```

The transition energy is set to zero.

zf

The number of possible final states (final state degeneracy).

Example:

```
zf: 0.0
```

The number of final states is set to zero.

q_energy_trans

The scattering vector connecting the two valleys in direct reciprocal coordinates.

Example:

```
q_energy_trans: [[0,0,0],[0.5,0.5,0.5]]
```

The scattering vector is set along the diagonal reciprocal cell.

Ionized impurity scattering parameters

Parameters using either the Conwell and Weisskopf (CW) or the Broks and Herring (BH) model to describe ionized impurity scattering.

n_i

The density of ionized impurities in units of $10^{21}\mathrm{cm}^{-3}$. Used for both the CW and BH model.

Example:

```
n_i: 0.01
```

The density of ionized impurities is set to $10^{19}\mathrm{cm}^{-3}$.

isl_i

The inverse screening length in units of inverse AA. Only used for the BH model.

Example:

```
isl_i: 0.3
```

The inverse screening length is set to 0.3 inverse AA.

z

The number of charge units of the impurity. In units of the electron charge.

Example:

```
z: 1.0
```

The charge of the impurity is set to one electron charge.

Alloy scattering parameters

A scattering model for the alloy $\mathrm{A}_x\mathrm{B}_{1-x}\mathrm{C}$.

vdiff

The atomic potential difference between the species A and B in eV.

Example:

```
vdiff: 1.0
```

The potential difference is set to 1.0 eV.

`alloyconc`

The concentration, x of the alloy.

Example:

```
alloyconc: 0.5
```

The concentration is set to 50%, i.e. 50% of A and 50% of B.

Common scattering parameters

Here follows scattering parameters that are shared between the different scattering mechnisms.

`eps`

The dielectric constant in units of the vacuum value.

Example:

```
eps: 12.0
```

The dielectric constant is set to 12.0 times the vacuum value.

`rho`

The mass density of the material in $\mathrm{g/cm^3}$.

Example:

```
rho: 2.4
```

The mass density of the material is set to 2.4 $\mathrm{g/cm^3}$.

`tau0_c`

The value of the constant relaxation time in units of fs.

Example:

```
tau0_c: 100.0
```

The constant relaxation time is set at 100.0 fs.

`emmission`

Determines if the considered scattering mechnism is by emmision or absorption. Acoustic phonon scattering includes both so this is only relevant where scattering of optical phonons is encountered.

Example:

```
emission: False
```

Use absorption, i.e. a phonon is absorbed in the scattering event.

Unit cell input parameters

- *Notes about format*
- *The unit cell*
  - *a*
  - *b*
  - *c*
- *Atomic positions*
  - *direct*
  - *pos*
  - *atomtypes*
- *K-point grid density*
  - *ksampling*

Notes about format

The input files follow normal YAML conventions. Please inspect the sample file `input/cellparam.yml`. Even though many parameters have default values if not specified the user should always run the calculations with fully specified input files for consistency and reproducibility.

The unit cell

`a`

The first lattice vector describing the unit cell in AA.

`b`

The second lattice vector describing the unit cell in AA.

`c`

The third lattice vector describing the unit cell in AA.

Example:

```
a: [5.0,0.0,0.0]
b: [0.0,5.0,0.0]
c: [0.0,0.0,5.0]
```

Generates a unit cell that is 5.0 by 5.0 by 5.0 AA.

Atomic positions

`direct`

Determines if the atomic positions are given in direct coordinates.

Example:

```
direct: True
```

The atomic positions are given in direct coordinates

`pos`

A list of the atomic positions. In direct or cartersian coordinates depending on the parameter `direct`.

Example:

```
pos: [[0.0,0.0,0.0]]
```

One atom centered at origo.

`atomtypes`

The type of atoms as a list in the same order as `pos`. Use abbreviations that are standard to the periodic table. An addition element X is added for unknown types.

Example:

```
atomtypes: [X]
```

K-point grid density

`ksampling`

The k-point sampling along each axis of the configured reciprocal unit cell.

Example:

```
ksampling: [15,15,15]
```

Use 15 by 15 by 15 samples in the full reciprocal unit cell.

6.6 Output

The transport coefficients are written to the *output* directory. The log file `info.log` is also written in this directory and can be monitored during a run to check the process.

6.6.1 Files

Electrical conductivity

The electrical conductivity can be found in the `sigma`. In units of $\mathrm{S/m}$. Consult the documentation in the header of the output file for layout.

Seebeck coefficients

The Seebeck coefficient can be found in the `seebeck`. In units of $\mu\mathrm{V/K}$. Consult the documentation in the header of the output file for layout.

Lorenz coefficient

The Lorenz coefficient can be found in the `lorenz`. In units of $10^{-8}\mathrm{V}^2/\mathrm{K}^2$. Consult the documentation in the header of the output file for layout.

The electrothermal conductivity

The electrical part of the thermal conductivity can be found in the `kappae`. In units of $\mathrm{W/mK}$.

The charge carrier concentration

The charge carrier concentration can be located in the `cc`. In units of $10^{21}\mathrm{cm}^{-3}$. Consult the documentation in the header of the output file for layout.

The Hall coefficient

The Hall coefficient (big R) can be located in the `hall`. In units of cm^3/C. Consult the documentation in the header of the output file for layout.

Warning: NOT YET IMPLEMENTED WHEN FIRST-PRINCIPLE INPUT IS UTILIZED (ONLY FILLED WITH BOGUS DATA). ONLY WORKS FOR SPHERICAL BANDS AT THE MOMENT.

The relaxation times

The total relaxation time for each band can be found in the files `scattering_band_n`, for band number *n*. In units of fs.

6.6.2 Visualization

Data can easily be visualized with Gnuplot. Currently no automatic visualization is performed. Data is blocked on temperature.

6.7 Tutorials

This document is currently empty, please consult the examples while the content is developed.

6.8 Examples

6.8.1 Silicon from first-principles

Here a short example of how to calculate the transport coefficients from a VASP output file (typically vasprun.xml) is presented.

Preparation

Start with the `param.yml`, `bandparam.yml` located in the *tests/13* directory. In the same directory a sample `vasprun.xml` file is also provided.

Copy these files into an *input* directory in the directory where you want to execute T4ME.

The general parameters

Here follows a brief explanation of the general parameters that need attention. They are already specified in the sample `param.yml` file so the user should not have to change this for a test run. All other parameters should at this point not need to be touched.

```
dispersion_interpolate: False
```

We do not want to interpolate the input at this step.

```
dispersion_velocities_numdiff: True
```

We need to calculate the group velocities of the electrons as this is not supplied by VASP by default.

```
transport_calc: True
```

Calculate the transport coefficients.

```
transport_method: "numerick"
```

Integrate numerically in k-space.

```
transport_integration_method: "trapz"
```

Used trapezoidal integration.

```
transport_chempot_min: -0.4
```

The minimum chemical potential to sample, in eV.

```
transport_chempot_max: 1.0
```

The maximum chemical potential to sample, in eV.

```
transport_chempot_samples: 20
```

How many samples do you want between `transport_chempot_min` and `transport_chempot_max`. Computational time and storage may vary depending on how this parameter is set.

```
transport_use_analytic_scattering: False
```

Use density-of-states models for the electron scattering.

```
dos_integrating_method: "trapz"
```

In order to calculate the scattering, it is necessary to calculate the density-of-states. Here this is done by using the trapezoidal integration with the delta function approximated by a Gaussian.

```
dos_smearing: 0.1
```

The smearing factor in eV used for the Gaussian approximation of the delta function. This needs to be sufficiently big in order for the density-of-states to converge, but also in order for the scattering data to have a smooth onset at the carrier energy. It is recommended that this, for calculations of scattering properties is not set below 0.1 eV.

```
temperature_min: 300
```

The minimum temperature in K.

```
temperature_max: 300
```

The maximum temperature in K.

```
temperature_steps: 1
```

The number of temperature steps between `temperature_min` and `temperature_max`.

```
e_fermi: True
```

Set the zero in energy to Fermi level supplied by the first-principle code, here VASP. The `transport_chempot_min` and `transport_chempot_max` parameters are thus set with reference to the shifted grid where the zero in energy is usually at the top valence band.

```
read: vasp
```

Read data from VASP output files, here `vasprun.xml`.

```
symprec: 1e-6
```

The symmetry cutoff used to detect symmetry. Should somewhat match with the value used in VASP. Passed along to Spglib to generate the irreducible to full Brillouin zone mapping.

```
onlytotalrate: True
```

Only store the total concatenated relaxation time arrays. Saves memory.

The band parameters

Here follows a brief explanation of the band parameters that need attention. They are already specified in the sample `bandparam.yml` file so the user should not have to change this for a test run. All other parameters should at this point not need to be touched.

```
Band 1-:
```

Tells the reader that it should apply all consecutive parameters to all the bands in the supplied system.

```
select_scattering: [0,0,0,0,0,0,0,0,0,0,0,1]
```

Only use constant scattering.

```
tau0_c: 100
```

The value of the constant relaxation time in fs.

Execution

After all parameters have been set (should only be necessary to copy files as stated before) the transport coefficients can be calculated by executing

```
python t4me.py
```

During execution the file `info.log` in the directory *output* can be inspected in order to assess progress and that everything works as expected.

Output

On completion the transport coefficients can be found in the *output* directory.

6.9 Custom interfaces

Here comes a description of how to customize new interfaces. In the meantime, please consult the structure and content of the `interfaces.py` file which among other contains interfaces to VASP which sets up the necessary data structure. This can be copied and modified to work against other codes.

6.10 API Documentation

6.10.1 scattering module

This module, `scattering.py` contains the setup and calculation of the scattering properties.

Contains routines to set up the scattering of the charge carriers.

`scattering.`**`check_scattering`**`(`*tr*`)`

Checks the scattering arrays.

Also that they are dimensionalized to the energy values stored in the current *Bandstructure()* object.

Parameters

tr [object] A *Transport()* object.

Returns

None

`scattering.`**`combined_scattering`**(*tr*, *energy*, *tau0*, *energy_trans*)

Calculates the total relaxation time.

Parameters

tr [object] A *Transport()* object

energy [float] The energy of the charge carrier in eV.

tau0 [ndarray]

Dimension: (12)

Contains the relaxation time prefactors for the different scattering mechanisms in units of fs.

energy_trans [ndarray]

Dimension: (12)

Contains the energy transitions in eV (that is added to the energy in
$tau =$
$tau_0 E^{r-1/2}$, typically, $E = E +$
$hbar$
$omega$, where
$hbar$
$omega$ is the size of the energy transition. Set it to zero for the non-relevant scattering mechanisms.

effmass [float] The effective mass in units of the electron mass

Returns

float The combined relaxation time in fs.

Notes

Calculates the total relaxation time

$$\begin{gathered} frac1 \\ tau = \\ sum_i \\ frac1 \\ tau_i, \end{gathered}$$

where
$tau =$
$tau_0 E^{r-1/2}$. The array *scattering_tau0_select* determines which scattering to include in the sum. Consult `scattering_parabolic()` for additional details. The scattering prefactors
tau_0 are ordered in a sequence described there. The *scattering_tau0_select* follows this sequence and is set in the bandstructure configuration file.

`scattering.`**`find_r_for_closed`**(*tr*, *band*)

Analyze the input tau0 and find the associated scattering values r.

Parameters

tr [object] A *Transport()* object

band [integer] The band index.

Returns

integer Two times the r value to avoid half integer values.

Notes

These are necessary for the analytic Fermi integrals.

`scattering.`**`interpolate`**(*tr*, *method='linear'*)

Interpolates the scattering array on all available energies.

Parameters

tr [object] A *Transport()* object containing the scattering arrays and the energies etc.

method [string, optional] The interpolation method to use. Uses the `interp1d()` function of Scipy and this sets the parameter *kind*. Defaults to "linear".

Returns

None

See also:

`scipy.interpolate.interp1d`

Notes

Here we only perform an interpolation on the array containing the total relaxation time since this is used during the transport calculations.

`scattering.`**`pad_scattering_values`**(*tr*)

Pad the scattering values.

Parameters

tr [object] A *Transport()* object.

Returns

None

Notes

The padded values are stored in the *tr* object.

We need to pad the energies where the dos is calculated with a larger number of samples such that we cover the whole energy range in the stored bandstructure due to later interpolation routines etc. not going out of bounds when such energies are passed to the interpolator etc.

We can set it to a large number because 1 eV outside the chemical potential. We already know no states contribute at temperatures below 2000 K.

`scattering.`**`scattering_dos`**(*tr*, *dos*, *energies*, *select_scattering*)

Setup scattering mechnisms.

Store values in the scattering arrays using the density of states data as the energy dependency.

Parameters

tr [object] A *Transport()* object

dos [ndarray]

Dimension: (N,M)

Array containing the partial density of states (1/eV/AA^3), where N is the band index and M is the energy index.

energies [ndarray]

Dimension: (M)

Array containing the energy in eV at M samplings where the density of states is calculated.

select_scattering [ndarray]

Dimension: (12)

Array containing integers. Set to 1 to select the scattering, 0 to exclude. The variables in *select_scattering* are set in the bandstructure configuration file, one value for each scattering and band. See notes below for the currrently available scattering mechnisms.

Returns

scattering_inv [ndarray]

Dimension: (T,N,M,12)

The scattering array in fs units in the current *Transport()* object for T temperature steps, N number of bands, M number of energy steps and 12 number of scattering mechanisms

scattering_total_inv [ndarray]

Dimension: (T, N, M)

The total (all mechanisms summed) scattering array in fs units) in the current *Transport()* object for T temperature steps, N number of bands and M number of energy steps

scattering_tau0 [ndarray]

Dimension: (T, N, 12)

The scattering prefactor array, tau0 in units of fs, in the current *Transport()* object for T temperature steps, N number of bands and 12 number of scattering mechanisms.

Notes

Currently only the following scattering mechanisms are supported:

select_scattering index	scattering mechanism
1	Acoustic phonon scattering from def. pot.
2	Non-polar optical phonon scattering from def. pot. (alpha stage)
3	Intervalley phonon scattering (alpha stage)
4	Polar optical phonon scattering (alpha stage)
5	None
6	None
7	None
8	None
9	None
10	None
11	None
12	Constant (energy and k-point independent)

Only the acoustic phonon scattering has been tested.

Consult the bandstructure configuration file for the respective constants that have to be set besides *select_scattering* and their units.

Todo: Add more extensive documentation for the different scattering mechanisms.

Warning: The scattering models based on density of states does currently not properly involve the energy shift required for the transfer energies. This is quite serious, but does not influene the acoustic phonon scattering (taken to be zero in the model implemented here). The current approach if not using the analytic parabolic models is that the transfer energies for all scattering mechanisms are summed and the energy of where the relaxation time is evaluated is shifted by this amount during interpolation. The sum approximation is not physically justified and needs additional investigation, also the interpolation are sensitive and can fail close to the van Hove singularities since the relaxation time is propotional to the inverse of the density of states. All known problems pertaining to density of states are also manifested here for the scattering.

`scattering.`**`scattering_parabolic`**(*tr*, *energies*, *select_scattering*, *use_eonk=False*)

Setup scattering mechnisms.

Store values in the scattering arrays using parabolic band dispersions as an approximation.

Parameters

tr [object] A *Transport()* object

energies [ndarray]

Dimension: (N)

Array containing the energy in eV at N samplings where the scattering values are o be calculated.

select_scattering [ndarray]

Dimension: (12)

Array containing integers. Set to 1 to select the scattering, 0 to exclude. The variables in *select_scattering* are set in the bandstructure configuration file, one value for each scattering and band. See notes below for the currrently available scattering mechnisms.

use_eonk [boolean] If set to True, generate the scattering values on the supplied energy for each band and on its k-points

Returns

scattering_inv [ndarray]

Dimension: (T,N,M,12)

The scattering array in fs units in the current *Transport()* object for T temperature steps, N number of bands, M number of energy steps and 12 number of scattering mechanisms

scattering_total_inv [ndarray]

Dimension: (T, N, M)

The total (all mechanisms summed) scattering array in fs units) in the current *Transport()* object for T temperature steps, N number of bands and M number of energy steps

scattering_tau0 [ndarray]

Dimension: (T, N, 12)

The scattering prefactor array, tau0 in units of fs, in the current *Transport()* object for T temperature steps, N number of bands and 12 number of scattering mechanisms.

Notes

Currently only the following scattering mechanisms are supported:

select_scattering index	scattering mechanism
1	Acoustic phonon scattering from def. pot.
2	Non-polar optical phonon scattering from def. pot.
3	Intervalley phonon scattering
4	Polar optical phonon scattering
5	Piezoelectric acoustic phonon scattering
6	Ionized impurity scattering, Brooks-Herring
7	Ionized impurity scattering, Conwell-Weisskopf
8	Alloy scattering
9	None
10	None
11	None
12	Constant (energy and k-point independent)

Also consult the bandstructure configuration file for the respective constants that have to be set besides *select_scattering* and their units.

Todo: Add more extensive documentation for the different scattering mechanisms.

6.10.2 t4me module

This module, `t4me.py` contains the main driver for the program. Most standard calls are included, but if the users wants, custom execution can be adapted in this file.

It is with this file T4ME is executed with the command

```
python t4me.py
```

T4ME - Transport 4 MatErials - a code to calculate the electronic transport coefficients of materials.

6.10.3 transport module

This module, `transport.py` contains the setup and checks necessary to execute the calculation of the transport coefficients. It also execute those routines.

Contains routines to set up the calculation of the charge carrier transport coefficients.

class `transport.`**`Transport`**(*bs*)

Bases: `object`

Involves all transport related routines.

Parameters

bs [object] A *Band()* object.

lattice [object] A *Lattice()* object.

param [object] A *Param()* object.

`calc_carrier_concentration`(*self*, *temperature*, *chempot*, *dos=None*, *dos_energies=None*, *band_decomp=False*, *defect_ionization=False*)

Returns the charge carrier concentration.

Parameters

temperature [float] The temperature in K.

chempot [float] The chemical potential in eV.

dos [ndarray, optional]

Dimension: (N,M)

Contains the band decomposed density of states for each band N and energy M. If not supplied, set to the *dos_partial* parameter of the current *Bandstructure()* object.

dos_energies [ndarray, optional]

Dimension: (M)

The energies in eV where the density of states are sampled.

band_decomp [boolean] Return a band decomposed carrier concentration or not.

defect_ionization [boolean] Selects if defect ionization compensation should be included. The *donor_number*, *donor_energy*, *donor_degen_fact*, *acceptor_number*, *acceptor_energy* and *acceptor_degen_fact* need to be set in the general configuration file.

Returns

n_type [ndarray]

Dimension: (N)

Contains the n-type carrier concentration for each band index N in units of $10^{21}\mathrm{cm}^{-3}$.

p_type [ndarray]

Dimension: (N)

Contains the p-type carrier concentration for each band index N in units of $10^{21}\mathrm{cm}^{-3}$.

`calc_transport_tensors`(*self*, *bs=None*, *temperatures=None*, *chempots=None*, *method=None*)
Selects which method to use when calculating the transport coefficients.

Parameters

bs [A *Band()* object containing the band structure.]

temperatures [ndarray, optional]

Dimension: (N)

Contains N different temperatures in K. If not supplied the *temperature* from the active *Transport()* object is used.

chempots [ndarray, optional]

Dimension: (M)

Contains M different chemical potentials in eV. If not supplied the *chempot* from the active *Transport()* object is used.

method [{"closed", "numeric", "numerick"}] If *method* is not supplied is defaults to "numeric" unless bandstructure data is read numerically or generated (all cases where the closed Fermi Dirac integrals cannot be used) when it defaults to "numerick".

"closed" evaluates the closed Fermi integrals where only
one scattering mechanism is possible per band. Only valid
for systems where one can strictly rely on a parametrized
parabolic bandstructure based on effective mass models.
Parameters (e.g. effective masses for each band) are set
in the bandstructure configuration file.
The driver routine is `lbtecoeff.parabolic_closed()`

"numeric" similar to "closed, but evaluates the Fermi
integrals in an open form (e.g. it is possible to
concatenate the scattering mechanisms, which is not
possible for the closed Fermi integrals).
The driver routine is `lbtecoeff.parabolic_numeric()`

"numerick" evaluates the transport integrals more generally
as an integral over the k-points. It is less restrictive
than the two other options, but also more prone to
convergence issues etc. However, for bandstructures
read from datafiles, this is the only option.
The driver routine is `lbtecoeff.numerick()`

Returns

sigma, seebeck, lorenz [ndarray, ndarray, ndarray]

Dimension: (N,M,3,3), (N,M,3,3), (N,M,3,3)

Returns the electrical condcutivity, Seebeck coefficient and Lorenz tensor for N temperature and M chemical potential steps in units of

$mathrmS/$
$mathrmm,$
mu
$mathrmV/$
$mathrmK,$
$mathrmV^2/$
$mathrmK^2$. These are stored in the current *Transport()* object.

See also:

lbtecoeff.parabolic_closed

lbtecoeff.parabolic_numeric

lbtecoeff.numerick

fetch_chempots (*self*, *store=True*)
Set up the chemical potential.

Parameters

store [boolean, optional] If given and set to True, the chempot array is in addition to being returned also stored in the current *Transport()* object.

Returns

chempot [ndarray]

Dimension: (N)

Contains N chemical potential linear samplings in units of eV. The parameters *transport_chempot_min*, *transport_chempot_max* and *transport_chempot_samples* in param.yml set the maximum and minimum chemical potential and its number of samples.

fetch_etas (*self*, *chempot*, *temperature*)
Calculate the reduced chemical potential

Parameters

chempot [ndarray]

Dimension: (N)

Contains N samples of the chemical potential in units of eV.

temperature [float] The temperature in K.

Returns

eta [ndarray]

Dimension: (N)

Contains N samples of the reduced chemical potential

fetch_relevant_bands (*self*, *tr=None*)
Locate bands that will be included in the transport integrals.

Parameters

tr [object, optional] A *Transport()* object.

Returns

None

Notes

The included bands are located by considering the input range of chemical potentials from *transport_chempot_min* and *transport_chempot_max* padded with the value *transport_energycutband* on each side (see the general configuration file).

fetch_temperatures (*self*, *store=True*)
Set up the temperatures.

Parameters

store [boolean, optional] If given and set to True, the temperature array is in addition to being returned also stored in the active *Transport()* object.

Returns

temperature [(N) ndarray] Contains N temperature linear samplings in units of K. The parameters *temperature_min*, *temperature_max* and *temperature_steps* in param.yml set the maximum and minimum temperature and its number of steps.

setup_scattering (*self*, *dos=None*, *dos_energies=None*, *select_scattering=None*)
Selects which how to set up the carrier scattering.

Parameters

dos [ndarray]

Dimension: (N,M)

Array containing the partial density of states in units of 1/eV/AA^3, where N is the band index and M is the energy index.

dos_energies [ndarray]

Dimension: (M)

Array containing the energy in eV at M samplings where the density of states is calculated.

select_scattering [ndarray]

Dimension: (12)

Array containing integers. Set to 1 to select the scattering, 0 to exclude. The variables in *select_scattering* are set in the bandstructure configuration file, one value for each scattering and band. See notes below for the currrently available scattering mechnisms.

Returns

None

See also:

scattering.scattering_dos

scattering.scattering_parabolic

transport.**acceptor_ionization** (*number*, *energy*, *degen*, *e_fermi*, *beta*)
Returns the number of ionized acceptors.

Parameters

number [float] Number of acceptors.

energy [float] The energy in eV where the acceptor compensation is to be evaluated.

degen [float] The acceptor degeneration number.

e_fermi [float] The Fermi level in eV.

beta [float] The beta (1/kT) factor in eV.

Returns

float The acceptor ionization compensation.

`transport.`**`donor_ionization`**(*number*, *energy*, *degen*, *e_fermi*, *beta*)

Returns the number of ionized donors.

Parameters

number [float] Number of donors.

energy [float] The energy in eV where the donor compensation is to be evaluated.

degen [float] The donor degeneration number.

e_fermi [float] The Fermi level in eV.

beta [float The beta (1/kT) factor in eV.]

Returns

float The donor ionization compensation.

`transport.`**`fermi_dist`**(*e*, *e_fermi*, *beta*)

Returns the Fermi Dirac distribution function (without spin degeneracy).

Parameters

e [float] The energy in eV where the Fermi Dirac distribution is to be evaluated.

e_fermi [float] The Fermi level in eV.

beta [float] The beta factor (1/kT) in eV.

Returns

float The value of the Fermi function (without spin degeneracy).

`transport.`**`fetch_chempot_from_etas`**(*temperature*, *etas*)

Calculate the chemical potential from eta and the temperature.

Parameters

temperature [float] The temperature in K.

etas [ndarray]

Dimension: N

The unitless chemical potential,
eta for N steps.

Returns

chempots [ndarray]

Dimension: N

The chemical potentials in units of eV.

6.10.4 lbteint module

This module, `lbteint.py` contains the integral solvers for the linearized Boltzmann Transport equations.

Contains the routines to perform the Boltzmann transport integrals.

`lbteint.`**`analytic_k_space_energy`**(*kx*, *ky*, *kz*, *effmass*, *e_shift*)
Returns the parabolic energy dispersion.

Parameters

transport [object] A *Transport()* object

kx [float] The k_x in cartesian coordinates.

ky [float] The k_y in cartesian coordinates.

kz [float] The k_z in cartesian coordinates.

effmass [ndarray]

Dimension: (3)

The effective mass along k_x, k_y and k_z, respectively.

Returns

float The energy value in eV.

Warning: This routine only accepts the diagonal elements of the effective mass tensor

`lbteint.`**`analytic_k_space_integrand`**(*kz*, *ky*, *kx*, *eta*, *beta*, *effmass*, *e0*, *i*, *l*, *m*)
Returns the integrand for the anlytic reciprocal space integration of the transport tensor.

Parameters

kz [float] The k_z in cartesian coordinates.

ky [float] The k_y in cartesian coordinates.

kx [float] The k_x in cartesian coordinates.

eta [float] The reduced chemical potential.

beta [float] The
$beta$ factor,
$beta = (k_b T)^{-1}$ in eV.

effmass [float] The effective mass in units of the free electron mass.

e0 [float] The energy shift, e.g. $E =$
$hbar^2 k^2/2m + E_0$, where E_0 is the energy shift in eV.

i [int] The order of the transport tensor.

l [{0,1,2}] The first index of the transport tensor.

m [{0,1,2}] The second index of the transport tensor.

Returns

float The integrand value.

`lbteint.`**`analytic_k_space_velocity`**(*kx*, *ky*, *kz*, *effmass*, *i*)
Returns the parabolic velocity dispersion.

Parameters

kx [float] The k_x in cartesian coordinates.

ky [float] The k_y in cartesian coordinates.

kz [float] The k_z in cartesian coordinates.

effmass [ndarray]

Dimension: (3)

The effective mass along k_x, k_y and k_z, respectively.

i [{0,1,2}] The direction to evaluate the velocity (0 is along k_x etc.).

Returns

float The velocity in eVAA.

Warning: This routine only accepts the diagonal elements of the effective mass tensor. The $hbar/m_e$ factor is not returned and need to be introduced externally.

`lbteint.`**`concatenate_integrand`**`(`*energies*, *velocities*, *scatter*, *spin_fact*, *chempot*, *beta*, *order*`)`
Concatenates the integrand in the Boltzmann transport integral.

`lbteint.`**`concatenate_integrand_band`**`(`*energies*, *velocities*, *tau*, *spin_fact*, *chempot*, *beta*, *order*`)`
Concatenates the integrand in the Boltzmann transport integral and sums the bands.

`lbteint.`**`fermiintclosed`**`(`*order*, *eta*, *spin_fact*`)`
Returns the value of the closed expressions for the Fermi integrals.

`lbteint.`**`integrandpar`**`(`*eps*, *transport*, *w0*, *eta*, *beta*, *energy_trans*, *effmass*, *i*`)`
Returns the integrand used in the analytic energy integration of the transport coefficients in `integrate_e()`

Parameters

eps [float] The reduced carrier energy.

transport [object] A *Transport()* object.

w0 [ndarray]

Dimension: (12)

Contains the scattering rate prefactor for the different scattering mechanisms in units of inverse fs.

eta [float] The reduced chemical potential.

beta [float] The
$beta$ factor in eV.

energy_trans [ndarray]

Dimension: (12)

Contains the energy transitions (that is added to the energy in
$tau =$
$tau_0 E^{r-1/2}$, typically, $E = E +$
$hbar$
$omega$, where
$hbar$

$omega$ is the size of the energy transition. Set it to zero for the non-relevant scattering mechanisms.

effmass [float] The effective mass in units of the electron mass.

i [int] The order of the transport integral to be evaluated.

Returns

float The integrand value.

Notes

The total scattering is calculated based on the well known scattering models for parabolic energy dispersions
$tau =$
tau_0
$epsilon^{r-1/2}$, where r is the scattering factor.

`lbteint.`**`integrandpardos`**(*eps*, *transport*, *w0*, *eta*, *beta*, *energy_trans*, *effmass*, *i*)
The integrand for the density of states integral over energy.

Parameters

eps [float] The reduced carrier energy

transport [object] A *Transport()* object

w0 [ndarray]

Dimension: (12)

Contains the scattering rate prefactor for the different scattering mechanisms in units of inverse fs. Not used in this routine, but it needs the dummy from the call argument.

eta [float] The reduced chemical potential

beta [float] The
$beta$ factor in eV. Not used in this routine, but it needs the dummy from the call argument.

energy_trans [ndarray]

Dimension: (12)

Contains the energy transitions (that is added to the energy in
$tau =$
$tau_0E^{r-1/2}$, typically, $E = E +$
$hbar$
$omega$, where
$hbar$
$omega$ is the size of the energy transition. Set it to zero for the non-relevant scattering mechanisms. Not used in this routine, but it needs the dummy from the call argument.

effmass [float] The effective mass in units of the electron mass. Not used in this routine, but it needs the dummy from the call argument.

i [int] The order of the transport integral to be evaluated. Not used in this routine, but it needs the dummy from the call argument.

Returns

float The integrand value for the density of states.

Notes

Calculates the density of states integrand

$$\begin{gathered} frac \\ epsilon^{1/2} 1+ \\ exp(\\ epsilon- \\ eta) \end{gathered}$$

`lbteint.`**`integrandpart2`**(*eps*, *transport*, *w0*, *eta*, *beta*, *energy_trans*, *effmass*, *i*)

Returns the integrand used in the analytic energy integration of the transport distribution function with a quadratic
tau term

Parameters

eps [float] The reduced carrier energy.

transport [object] A *Transport()* object.

w0 [ndarray]

Dimension: (12)

Contains the scattering rate prefactor for the different scattering mechanisms in units of inverse fs.

eta [float] The reduced chemical potential.

beta [float] The
$beta$ factor in eV.

energy_trans [ndarray]

Dimension: (12)

Contains the energy transitions (that is added to the energy in
$tau =$
$tau_0 E^{r-1/2}$, typically, $E = E +$
$hbar$
$omega$, where
$hbar$
$omega$ is the size of the energy transition. Set it to zero for the non-relevant scattering mechanisms.

effmass [float] The effective mass in units of the electron mass.

i [int] The order of the transport integral to be evaluated.

Returns

float The integrand value.

Notes

The total scattering is calculated based on the well known scattering models for parabolic energy dispersions
$tau =$

tau_0
$epsilon^{r-1/2}$, where r is the scattering factor. Difference from `integrandpar()`: here tau^2 is used in the integrand (for the calculation of the Hall factor).

`lbteint.`**`scipy_e_integrals`**(*transport*, *integrand*, *e_min*, *e_max*, *w0*, *eta*, *beta*, *energy_trans*, *effmass*, *order*, *spin_fact*, *method='quad'*)

Calculates the one dimensional energy integrals.

Uses the SciPy function `scipy.integrate.quad()`.

Parameters

transport [object] A *Transport()* object

integrand [{"normal","hall","dos"}] Selects the type of integrand to be used. "normal" selects `integrandpar()`. "hall" selects `integrandpart2()`, "dos" selects `integrandpardos()`.

e_min [float] The lower integration limit in eV.

e_max [float] The higher integration limit in eV.

w0 [ndarray]

Dimension: (12)

Contains the scattering rate prefactor (inverse of relaxation time) for the different scattering mechanisms in units of inverse fs.

eta [float] The reduced chemical potential.

beta [float] The
$beta$ factor, (
$mathrmk_bT)^{-1}$ in eV.

effmass [ndarray]

Dimension: (3)

The effective mass along the three reciprocal unit vectors in units of the free electron mass.

e0 [float] The energy shift, e.g. $E =$
$hbar^2k^2/2m + E_0$, where E_0 is the energy shift in eV.

i [int] The order of the transport tensor.

l [{0,1,2}] The first index of the transport tensor.

m [{0,1,2}] The second index of the transport tensor.

spin_fact [int] The spin degeneracy factor. 1 for non-spin degeneracy, 2 for spin degeneracy.

method [{"quad"}, optional] The SciPy three dimensional integration method using `scipy.integrate.quad()`.

Returns

float The resulting integral over energy.

`lbteint.`**`scipy_k_integrals`**(*eta*, *beta*, *effmass*, *e0*, *i*, *l*, *m*, *method='tplquad'*)

Calculates the three dimensional wave vector integrals.

Uses the SciPy function `scipy.integrate.tplquad()`.

Parameters

eta [float] The reduced chemical potential

beta [float] The
$beta$ factor, (
$mathrmk_bT)^{-1}$ in eV.

effmass [ndarray]

Dimension: (3)

The effective mass along the three reciprocal unit vectors in units of the free electron mass.

e0 [float] The energy shift, e.g. $E = hbar^2k^2/2m + E_0$, where E_0 is the energy shift in eV.

i [int] The order of the transport tensor.

l [{0,1,2}] The first index of the transport tensor

m [{0,1,2}] The second index of the transport tensor

method [{"tplquad"}, optional] The SciPy three dimensional integration method using `scipy.integrate.tplquad()`.

Returns

float The resulting integral over the wave vectors.

See also:

`scipy.integrate.tplquad`

`lbteint.`**`scipy_k_integrals_discrete`**(*tr*, *integrand_type*, *energies*, *velocities*, *scatter*, *chempot*, *beta*, *order*, *spin_fact*, *method='trapz'*)

Calculates the three dimensional integrals over the k-points for discrete data.

Uses SciPy integration functions for discrete data.

Parameters

tr [object] A *Transport()* object

energies: ndarray Contains the band energies in eV for each k-point.

velocities: ndarray Contains the derivative if *energies* without the *hbar* factors for each k-point.

scatter: Contains the relaxation time at each k-point.

chempot [float] The chemical potential in eV

beta [float] The
$beta$ factor, (
$mathrmk_bT)^{-1}$ in eV.

spin_fact [int] The spin factor, 1 for non-spin degeneracy and 2 for spin degeneracy.

method [{"trapz", "simps", "romb"}, optional] The SciPy three dimensional integration method for the `scipy.integrate.trapz()`, `scipy.integrate.simps()` and the `scipy.integrate.romb()` functions, respectively. Defaults to "trapz".

Returns

integral [float] The resulting integral over the wave vectors.

`lbteint.`**`scipy_k_integrals_discrete2`**(*tr*, *energies*, *velocities*, *scatter*, *chempot*, *beta*, *spin_fact*, *order*, *method='trapz'*)

Calculates the three dimensional integrals over the k-points for discrete data.

Uses integration functions for discrete data.

Parameters

tr [object] A *Transport()* object

chempot [float] The chemical potential in eV

beta [float] The
$beta$ factor, (
$mathrmk_bT)^{-1}$ in eV.

spin_fact [int] The spin factor, 1 for non-spin degeneracy and 2 for spin degeneracy.

kx, ky, kz [float, float, float] The spacing in inverse AA between the points along each direction.

order [float] The order of the energy minus chemical potential term in the denominator.

method [{"trapz", "simps", "romb"}, optional] The SciPy three dimensional integration method for the `scipy.integrate.trapz()`, `scipy.integrate.simps()` and the `scipy.integrate.romb()` functions, respectively. Defaults to "trapz".

6.10.5 lbtecoeff module

This module, `lbtecoeff.py` contains the setup and calls to the integral solvers and possible also external integral solvers such that the linearlized Boltzmann Transport Equations might be solved.

Contains routines that sets up and selects the integral functions to be called.

`lbtecoeff.`**`calculate_hall_carrier_concentration`**(*hall*)

Calculates the Hall carrier concentration $n_h = ($
$mathrmeR_h)^{-1}$.

Parameters

hall [ndarray]

Dimension: (N,M,3,3)

The Hall tensor R_h in units of
$mathrmcm^3/$
$mathrmC$ for N temperature and M chemical potential samplings.

Returns

ndarray

Dimension: (N,M,3,3)

The Hall carrier concentration in units of $10^{-21}\mathrm{cm}^{-3}$

`lbtecoeff.`**`calculate_hall_factor`**(*n*, *nh*)

Calculates the Hall factor $r_h = n/n_h$

Parameters

n [ndarray]

Dimension: (N,M,3,3)

The calcalated carrier concentration in for N temperature and M chemical potential samplings in units of 10^{21}
$mathrmcm^{-3}$.

n [ndarray]

Dimension: (N,M,3,3)

The Hall carrier concentration in for N temperature and M chemical potential samplings in units of 10^{21}
$mathrmcm^{-3}$.

Returns

ndarray

Dimension: (N,M,3,3)

The Hall factor.

`lbtecoeff.`**`numerick`**`(`*tr*, *chempots*, *temperatures*, *bs=None*`)`

Calculates the transport coefficients according to the tensors defined in `full_k_space_analytic()`

Parameters

tr [object] A *Transport()* object.

chempots [ndarray]

Dimension: (N)

The N chemical potentials in eV on which to calculate the transport coefficients.

temperature [float]

Dimension: (M)

The M temperatutes in K.

bs [ndarray, optional]

Dimension: (I,J)

The energy dispersion in eV for the charge carriers for I bands and J k-points. If not given, it defaults to the *Bandstructure()* object stored in *tr*.

Returns

sigma, seebeck, lorenz [ndarray, ndarray, ndarray]

Dimension: (M,N,3,3)

Returns the electrical condcutivity, Seebeck coefficient and Lorenz tensor for M temperature and N chemical potential steps in units of
$mathrmS/$
$mathrmm$,
mu
$mathrmV/$
$mathrmK$,
$mathrmV^2/$
$mathrmK^2$.

Notes

This routine accepts a predetermined array containg the relaxation time sampled at different carrier energy steps. When the evaluation of the intergrals are executed the carrier energy dispersion, velocity and scattering array are interpolated on all energies stored in *bs* and then the integrals are evaluated statically (if *transport_integration_method* is “trapz”, “simps”, “romb”, “tetra” or “smeared”). The former method can also be used in the case where the scattering array is pretetermined by setting *transport_use_scattering_ontfly* in the general configuration file to True.

`lbtecoeff.`**`parabolic_closed`**(*tr*, *eta*, *bs*, *tau0_t*, *temperature*)

Calculates the parabolic Fermi integrals for the transport coefficients.

Parameters

tr [object] A *Transport()* object

eta [float] The reduced chemical potential

bs [object] A *Bandstructure()* object containing the dispersion relations of the N included bands

tau0_t [(N,M) ndarray] The relaxation time approximation (RTA) prefactors (tau0) in units of fs for the N bands and M defined scattering mechanisms. This is converted to the r factor in order to use the closed Fermi integrals from tau0

temperature [float] The temperature in K.

Returns

tupple [ndarray, ndarray, ndarray, ndarray, ndarray, ndarray]

Dimension: (3,3), (3,3), (3,3), (3,3), (3,3), (3,3)

The electrical conductivity, Seebeck coefficient, Lorenz number, Hall coefficient (big R, where the small Hall factor is divided by the charge carrier concentration) and charge carrier concentration for n and ptype carriers in units of

$$\begin{aligned}
mathrmS/\\
mathrmm,\\
mu\\
mathrmV/\\
mathrmK, 10^{-8}\\
mathrmV^2/\\
mathrmK^2,\\
mathrmcm^3/\\
mathrmC, 10^{21}\\
mathrmcm^{-3}, respectively.
\end{aligned}$$

See also:

`setup_scattering`

`find_r_from_tau0`

`parabolic_numeric`

Notes

The closed equations for the Fermi integrals can easily be developed from the following equaion from `parabolic_numeric()`

$$
\begin{aligned}
Sigma^i_{lm} = -\\
frac4e^2\\
sqrtm3\\
sqrt2\\
pi^2\\
hbar^3\\
int\\
tau(E)E^{3/2}(E-\\
mu)^i\\
frac\\
partial f_0\\
partial EdE.
\end{aligned}
$$

We now chose the i for the given tensor and enforce a scattering model of the form
$tau =$
$tau_0 E^{r-1/2}$, where
tau_0 is constant in energy. Finally we use the product rule and expand the integral. This yields

$$
\begin{aligned}
Sigma^0 =\\
frac4e^2\\
sqrtm3\\
sqrt2\\
pi^2\\
hbar^3(r+1)F_r(E),
\end{aligned}
$$

$$
\begin{aligned}
Sigma^1 =\\
frack_n e_n\\
left(\\
frac(r+2)F_{r+1}(E)(r+1)F_r(E)-\\
mu\\
right),
\end{aligned}
$$

$$Sigma^2 =$$
$$left($$
$$frack_n e_n$$
$$right)^2$$
$$left($$
$$frac(r+3)F_{r+2}(E)(r+1)F_r(E)-$$
$$frac(r+2)F_{r+1}(E)(r+1)F_r(E)$$
$$right),$$

where $F_i(E)$ is the famous Fermi integrals

$$F_i(E) =$$
$$int f_0 E^i dE.$$

It is customary to introduce the dimensionless energy
$epsilon = E/k_bT$ and chemical potential
$eta =$
mu/k_bT. We have the general relation

$$F_i(E) =$$
$$beta^{-(i+1)}F_i($$
$$epsilon),$$

and the
$Sigma$ coefficients can be easily transformed. Please consult `parabolic_numeric()` for the specific transport tensors and units used.

The execution of this method needs currently needs the effective mass tensor to be isotropic and only one scattering mechanism can be used per band. This method is selected by setting *transport_method* to "closed" in the general configuration file.

lbtecoeff.**parabolic_numeric**(*tr*, *eta*, *bs*, *tau0_t*, *temperature*)

The solution of the energy integrals for the BTE RTA for a parabolic dispersion.

Parameters

tr [object] A *Transport()* object.

eta [float] The reduced chemical potential.

bs [object] A *Bandstructure()* object containing the energy dispersions for the N bands.

tau0_t [(N,M) ndarray] The relaxation time approximation (RTA) prefactors (tau0) in units of fs for the N bands and M defined scattering mechanisms.

temperature [float] The temperature in K.

Returns

tupple, ndarray, ndarray, ndarray, ndarray, ndarray

Dimension: (3,3), (3,3), (3,3), (3,3), (3,3), (3,3)

The electrical conductivity, Seebeck coefficient, Lorenz number, Hall coefficient (big R, where the small Hall factor is divided by the charge carrier concentration) and charge carrier concentration for n and ptype carriers in units of

$$S/m,$$
$$mu\mathrm{V/K}, 10^{-8}\mathrm{V}^2/\mathrm{K}^2, \mathrm{cm}^3/\mathrm{C}, 10^{21}\mathrm{cm}^{-3}`,$$

respectively.

Notes

This routine is the same as `parabolic_closed()` except here we solve the Fermi intergrals numerically. This allows to use concatenated scattering mechanisms for each band. Otherwise the approximations and requirements are similar. The method is selected by setting *transport_method* to “numeric”.

The transport coefficients are defined as

$$
\begin{aligned}
Sigma^i_{lm} = -s \\
frace^2 8 \\
pi^3 \\
int \\
tau(E(\\
veck))v_l(E(\\
veck))v_m(E(\\
veck))(E(\\
veck)- \\
mu)^i \\
frac \\
partial f_0 \\
partial E(\\
veck)d \\
veck.
\end{aligned}
$$

Using the fact that d
$veck = k^2$
$sin($
$theta)d$
$thetad$

phi we get

$$\begin{aligned}
Sigma^i_{lm} = -s\\
frace^2 2\\
pi^2\\
intk^2\\
tau(E(\\
veck))v_l(E(\\
veck))v_m(E(\\
veck))(E(\\
veck)-\\
mu)^i\\
frac\\
partial f_0\\
partial E(\\
veck)dk.
\end{aligned}$$

For parabolic bands $E =$
$hbar^2k^2/2m$. We also use $E = mv^2/2$. Furthermore we assume that our crystal is isotropic and cubic, such that
$Sigma =$
$Sigma_{00}$. Then $v_i^2 = (v_{xx} + v_{yy} + v_{zz})/3 = v^2/3$ and the expression above simplifies to

$$\begin{aligned}
Sigma^i = -s\\
frac\\
sqrt2e^2\\
sqrtm3\\
pi^2\\
hbar^3\\
int\\
tau(E)E^{3/2}(E-\\
mu)^i\\
frac\\
partial f_0\\
partial EdE
\end{aligned}$$

And we have a very manageble integral over energy. Here, we have assumed that
tau can be expressed in terms of energy instead of the wave vector. A similar procedure can be used to obtain energy integrals for other dispersion relations than parabolic. As opposed to `parabolic_closed()` we here want to solve the integrals numerically (in order to be able to use composite

tau) and thus want to simplify the mathematical operations in the integrands as much as possible. Since

$$partial f_0/ partial E = - beta/(2(1+ cosh(beta(E- mu)))),$$

where
$beta = (k_bT)^{-1}$ we get

$$Sigma^i = s frac sqrt2me^2 beta6 pi^2 hbar^3 int frac tau(E)E^{3/2}(E- mu)^i 1+ cosh(beta(E- mu))dE$$

Notice now that there is a factor of 1/2 difference in front of these integrals compared to the ones in `parabolic_closed()` due to the expansion of the derivative of the Fermi function with respect to energy. The factor $s = 2$ accounts for spin degeneracy, otherwise $s = 1$ and is set to True or False for each band with the parameter *spin_deg* in the bandstructure configuration file.

It is customary to introduce the dimensionless energy
$epsilon = E/k_bT$ and chemical potential
$eta =$

mu/k_bT. Doing this we obtain

$$
\begin{aligned}
Sigma^i = s \\
frac \\
sqrt 2 \\
mathrm m \\
mathrm e^2 6 \\
mathrm \\
hbar^3 \\
pi^2 \\
beta^{i+3/2} \\
int \\
frac(\\
epsilon - \\
eta)^i \\
tau(\\
epsilon/ \\
beta) \\
epsilon^{3/2} 1+ \\
cosh(\\
epsilon - \\
eta) d \\
epsilon
\end{aligned}
$$

The carrier density n can be calculated using

$$
\begin{aligned}
n = s \\
frac 18 \\
pi^3 \\
int f_0 d \\
veck
\end{aligned}
$$

Depending on what kind of scattering models that is chosen, the rescaling of
tau needs to be performed for a particular scattering model in order to pull the correct
$beta$ factor outside the integral.

In order to obtain the correct units the equation above for each transport coefficient have been implemented as

Electrical conductivity:

$$
\begin{gathered}
fracs \\
sqrt206 \\
piG \\
frac \\
sqrtm_c\hbar_c \\
tilde\hbar k^{3/2}a^{1/2}T^{3/2} \\
int_0 \\
infty \\
frac \\
tau(\\
epsilon/ \\
beta) \\
epsilon^{3/2}1+ \\
cosh(\\
epsilon- \\
eta)d \\
epsilon \\
left[\\
frac \\
mathrmS \\
mathrmm \\
right].
\end{gathered}
$$

One usually set $s = 2$. Seebeck coefficient:

$$
\begin{aligned}
alpha = 10^2 \\
frack_n e_n \\
frac \\
int_0 \\
infty \\
frac \\
tau(\\
epsilon/ \\
beta)(\\
epsilon- \\
eta) \\
epsilon^{3/2} 1+ \\
cosh(\\
epsilon- \\
eta)d \\
epsilon \\
int_0 \\
infty \\
frac \\
tau(\\
epsilon/ \\
beta) \\
epsilon^{3/2} 1+ \\
cosh(\\
epsilon- \\
eta)d \\
epsilon \\
left[\\
frac \\
mu\mathrm{VK} \\
right].
\end{aligned}
$$

Lorenz number:

$L =$
$left($
$frack_ne_n$
$right)^2$
$left($
$frac$
int_0
$infty$
$frac$
$tau($
$epsilon/$
$beta)($
$epsilon-$
$eta)^2$
$epsilon^{3/2}1+$
$cosh($
$epsilon-$
$eta)d$
$epsilon$
int_0
$infty$
$frac$
$tau($
$epsilon/$
$beta)$
$epsilon^{3/2}1+$
$cosh($
$epsilon-$
$eta)d$
$epsilon -$
$left($
$frac$
int_0
$infty$
$frac$
$tau($
$epsilon/$
$beta)($
$epsilon-$
$eta)$
$epsilon^{3/2}1+$
$cosh($
$epsilon-$
$eta)d$
$epsilon$
int_0
$infty$

Hall coeffcient:

$$\begin{array}{r}
R_H = 10^{-3} \\
frac3 \\
pi^2 4 \\
sqrt5 \\
mathrme_n \\
left(\\
frac \\
mathrmm_c k \\
mathrm\hbar_c^2 \\
right)^{-3/2} \\
left(aT \\
right)^{-3/2} \\
frac \\
int_0 \\
infty \\
frac \\
tau(\\
epsilon/ \\
beta)^2 \\
epsilon^{3/2}1+ \\
cosh(\\
epsilon- \\
eta)d \\
epsilon \\
left(\\
int_0 \\
infty \\
frac \\
tau(\\
epsilon/ \\
beta) \\
epsilon^{3/2}1+ \\
cosh(\\
epsilon- \\
eta)d \\
epsilon \\
right)^2 \\
left[\\
frac \\
mathrmcm^3 \\
mathrmC \\
right].
\end{array}$$

Carrier concentration:

$$\begin{array}{r} n = \\ frac20 \\ sqrt5 \\ pi^2 \\ left(\\ frac \\ mathrmm_c k \\ mathrm\hbar_c^2 \\ right)^{3/2} \\ left(aT \\ right)^{3/2} \\ int_0 \\ infty \\ frac \\ sqrt \\ epsilon1+ \\ cosh(\\ epsilon- \\ eta)d \\ epsilon \\ left[10^{21} \\ mathrmcm^{-3} \\ right]. \end{array}$$

From these, the Hall carrier concentration n_h and Hall factor, r_h can be calculated

$$\begin{array}{r} n_h = \\ frac1eR_H, \end{array}$$

and

$$\begin{array}{r} r_h = \\ fracnn_h. \end{array}$$

The multiband expressions are accordingly (using i as the band index)

$$\begin{array}{r} n = \\ sum_i n_i \end{array}$$

$$\begin{array}{r} sigma = \\ sum_i \\ sigma_i \end{array}$$

$alpha =$
$frac$
sum_i
$alpha_i$
$sigma_i$
sum_i
$sigma_i$

$L =$
$frac$
$sum_i L_i$
$sigma_i$
sum_i
$sigma_i +$
$frac$
sum_i
$sigma_i$
$alpha_i^2$
sum_i
$sigma_i -$
$alpha^2$

The units on the rest of the variables are

tau [fs]
The relaxation time given in fs.

$T[$
mathrmK]
The temperature given in K.

a
The effective mass factor, e.g. $m^* = am_e$.

Furthermore, the coefficients that sets the correct scaling is defined as

$k = 8.6173324$
The coefficient of the Boltzmann constant.

$e_n = 2.417989348$
The coefficient for the normalized (e/h) electron charge.

$m_c = 0.510998928$
The coefficient of mc^2

$\hbar_c = 197.3269718$
The coefficient of

hbar c

tilde$\hbar = 6.582119514$
The coefficient of
hbar

:math::*G = 7.7480917346*
The coefficient of the conductance quantum

Warning: ONLY VALID FOR PARABOLIC ENERGY DISPERSIONS AND SCATTERING MODELS

Todo: ADD POSIBILITY TO USE DIFFERENT EFFECTIVE MASSES ALONG DIFFERENT DIRECTIONS

`lbtecoeff.`**`parabolice`**(*tr*, *eta*, *temperature*, *bs*, *tau0*, *method*)

A wrapper for all the parabolic Fermi integrals.

Parameters

tr [object] A *Transport()* object

eta [float] Contains the reduced chemical potential.

temperature [float] Contains the temperature in K.

bs [object] *Bandstructure()* object containing the band structure.

method [{“numeric”, “closed”}] How to evaluate the Fermi integrals.

“numeric”: solve the Fermi integrals using
numerical integration
“closed”: solve the closed Fermi integrals
(exact analytic expressions)

Returns

tupple: ndarray, ndarray, ndarray, ndarray, ndarray, nadarray

Dimension: (3,3), (3,3), (3,3), (3,3), (3,3), (3,3)

The electrical conductivity, Seebeck coefficient, Lorenz number, Hall coefficient (big R, where the small Hall factor is divided by the charge carrier concentration) and charge carrier concentration in units of

$$S/m,$$
$$mu\mathrm{V/K}, 10^{-8}\mathrm{V}^2/\mathrm{K}^2, \mathrm{cm}^3/\mathrm{C}, 10^{21}\mathrm{cm}^{-3}`,$$

respectively.

Notes

All integrals in this function is evaluated over energy and not k-points (one of the most usual procedures for solving the Boltzmann transport integrals)

6.10.6 lattice module

This module, `lattice.py` contains the setup and lattice related routines, including both the real and reciprocal part of it.

Contains routines to setup the lattice.

class `lattice.`**`Lattice`**(*param*, *location=None*, *filename=None*)

Bases: `object`

Contains routines to set up the system atmosphere.

This includes the unit cell, the BZ and IBZ k-point mesh etc.

Read the YAML cell configuration file (cellparams.yml by default is used to read basic cell parameters if for instance this is not set up by other inputs in the *`interface`*)

Parameters

param [object] A *Param()* object containing the parameters of the general configuration file.

class Kmesh

Bases: `object`

Data container for the k-point mesh generation.

Currently not used throughout the program, only for the intial setup.

Todo: Incorporate this class into the whole program.

`cart_to_dir`(*self*, *v*, *real=False*)

Calculates the direct vector if the input is a vector in cartesian coordinates.

Parameters

v: ndarray

Dimension: (3)

The input direct vector.

real: boolean If set to False, the reciprocal unitcell is used, is set to True, the unitcell is used.

Returns

ndarray

Dimension: (3)

The cartesian vector.

Notes

Typically the transformation in reciprocal space is

where
$veck$ and
$veck'$ is the reciprocal vector in direct and cartesian coordinate systems, respectively. Here, B is the reciprocal unit cell.

`check_for_duplicate_points` (*self*)
Checks for duplicate k-points. This is currently not supported.

Parameters

None

Returns

None

`check_lattice` (*self*)
Checks if the celldata is present and that the most important parameters are defined.

Parameters

None

Returns

None

`check_mesh` (*self*)
Checks that the most important parameters for the k-point mesh have been set.

Parameters

None

Returns

None

`create_kmesh` (*self, ksampling=None, shift=array([0, 0, 0], dtype=int32), halfscale=True, borderless=False*)
Returns the k point mesh.

Parameters

ksampling: ndarray, optional

Dimension: (3)

Contains the k - point sampling points along each direction. If not supplied the *ksampling* of the current *Lattice()* object is used.

shift: ndarray, optional

Dimension: (3)

Contains the shift for the k - point mesh generation. If not supplied the default is set to[0.0, 0.0, 0.0]

halfscale: boolean Selects if the BZ mesh should go from -0.5 to 0.5 or -1.0 to 1.0. If not supplied, the default is set to True.

borderless: boolean Selects if the BZ border points should be included in the mesh generation. True selects borderless, e.g. -0.5, 0.5 etc. are excluded.

Returns

mapping_bz_to_ibz: ndarray

Dimension: (N, 3)

Contains a mapping table such that it is possible to go from the BZ to the IBZ mesh. Stored in the current *Lattice()* object.

mapping_ibz_to_bz: ndarray

Dimension: (M, 3)

Contains a mapping table such that it is possible to go from the IBZ to the BZ mesh. Stored in the current *Lattice()* object.

kmesh: ndarray

Dimension: (N, 3)

The k - point mesh in the full BZ for N sampling points determined by the multiplication of the content of *ksampling*. Stored in the current *Lattice()* object.

kmesh_ibz: ndarray

Dimension: (M, 3)

The k - point mesh in the irreducible BZ. The number of points M is dependent on the symmetry. Usually M < N. Stored in the current *Lattice()* object.

ksampling: ndarray

Dimension: (3)

The full BZ k - point sampling in each direction. Stored in the current *Lattice()* object.

Notes

This routines use spglib, an excellent tool written by A. Togo.

`dir_to_cart`(*self*, *v*, *real=False*)

Calculates the cartesian vector if the input is a vector in direct coordinates.

Parameters

v: ndarray

Dimension: (3)

The supplied cartesian vector.

real: boolean If set to False, the reciprocal unitcell is used, is set to True, the unitcell is used.

Returns

ndarray

Dimension: (3)

The direct vector.

Notes

Typically the transformation in reciprocal space is

where
$veck$ and
$veck'$ is the reciprocal vector in direct and cartesian coordinate systems, respectively. Here, B is the reciprocal unit cell.

fetch_bz_border(*self*, *kmesh=None*, *direct=True*)
Returns the BZ border in direct or cartesian coordinates

Parameters

kmesh: ndarray, optional

Dimension: (N, 3)

The k - point mesh for N k - points. If not supplied, the value of the current *Lattice()* is used.

direct: boolean, optional Selects if direct coordinates are to be returned (True, default) or cartesian(False).

Returns

ndarray

Dimension: (3)

Contains the BZ border points(largest coordinate along each axis).

fetch_iksampling(*self*)
Fetches the a denser k-point sampling which is for instance used when one would like to interpolate

Parameters

None

Returns

iksampling [ndarray]

Dimension: (3)

The number of requested k-point sampling along each reciprocal unit vector.

fetch_kmesh(*self*, *direct=True*, *ired=False*)
Calculates the k point mesh in direct or cartersian coordinates.

Parameters

direct [boolean] Selects the k-point grid in direct (True) or cartesian coordinates (False)

ired [boolean] Selects the ireducible k-point grid (True), or the full grid (False)

Returns

ndarray

Dimension: (M,3)

Contains M k-points representing the k-point mesh.

fetch_kmesh_step_size(*self*, *direct=False*)
Returns the step size along each direction.

Parameters

direct [boolean, optional] If True the step size is returned in direct coordinates, otherwise it is returned in AA^{-1} units. Defaults to False.

Returns

stepx, stepy, stepz [float, float, float] The step size along each reciprocal lattice vector.

Notes

Regularly spaced and ordered grids are assumed. Also, the step size returned is with respect to the reciprocal unit cells unit vectors. If *direct* is True the step size between 0 and 1 is returned, while for False, this step size is scaled by the length of the reciprocal unit vectors in AA^{-1}.

`fetch_kmesh_unit_vecs`(*self*, *direct=True*)
Calculates the k-point mesh sampling points along the unit vectors.

Works in direct or cartesian coordinates.

Parameters

direct [boolean] Selects to return direct (True) or cartesian (False) unit vectors.

Returns

ndarray

Dimension: (3)

The unit vectors of the k-point mesh.

`fetch_kpoints_along_line`(*self*, *kstart*, *kend*, *stepping*, *direct=True*)
Calculates the k - points along a line in the full BZ k - point mesh

Parameters

kstart: ndarray

Dimension: (3)

The start k - point in direct coordinates.

kend: ndarray

Dimension: (3)

The end k - point in direct coordinates.

stepping: int The N number of steps along the line.

direct: boolean If True direct coordinates are returned, else cartesian coordinates are returned.

Returns

ndarray

Dimension: (N)

The N number of k - point cartesian coordinates along the line.

`fetch_ksampling_from_stepsize`(*self*, *step_sizes*)
Calculates the ksampling based on the step size.

Parameters

step_sizes [ndarray]

Dimension: (3)

The step size along each reciprocal unit vector

Returns

ksampling [float, float, float] The suggested sampling along each reciprocal unit vector

`fetch_length_runitcell_vecs`(*self*)

Returns the length of each reciprocal lattice vector.

Parameters

None

Returns

ndarray

Dimension: (3)

The lenght of each reciprocal lattice vector in inverse AA.

`generate_consistent_mesh`(*self*)

Calculates the k-point mesh and sets up proper mapping.

Also makes sure the IBZ or BZ supplied is similar to the one generated by spglib given the symmetry used.

Parameters

None

Returns

None

Notes

This should be reconsidered in the future as this is bound to give the user problems. Consider writing an interface which can accept the symmetry operators and use these directly.

`real_to_rec`(*self*, *unitcell=None*)

Calculates the reciprocal unitcell from the real unitcell.

Parameters

unitcell: ndarray, optional

Dimension: (3, 3)

A real unitcell. Defaults to the unitcell for the current *Lattice()* object.

Returns

ndarray

Dimension: (3, 3)

The reciprocal unitcell, with: math: *vec{b_1}* = [0][:] etc.

`rec_to_real`(*self*, *unitcell=None*)

Calculates the real unitcell from the reciprocal unitcell.

Parameters

unitcell: ndarray, optional

Dimension: (3, 3)

A reciprocal unitcell. Defaults to the internal runitcell for the current *Lattice()* object.

Returns

ndarray

Dimension: (3, 3)

The real unitcell, with: math: *\vec{a_1}* = [0][:] etc.

regularcell(*self*)

Checks that all the vectors in the unit cell is orthogonal and thus if the cell is regular.

Parameters

None

Returns

regular: boolean True if regular, False otherwise.

`lattice.`**calculate_cell_volume**(*cell*)

Calculates the cell volume.

Parameters

cell [ndarray]

Dimension: (3,3)

Contains the i basis vectors of the cell, [i,:].

Returns

volume [float] The volume of the cell in units of the units along the input axis cubed.

`lattice.`**check_sensible_ksampling**(*ksampling*)

Check if the ksampling is sensible.

Parameters

ksampling [ndarray]

Dimension: (3)

The k-point sampling along each reciprocal lattice vector.

Returns

None

6.10.7 bandstructure module

This module, `bandstructure.py` contains the setup and calculation of bandstructure related properties. This also include for instance the calculation of the density of states.

Contains routines to set up the bandstructure.

class `bandstructure.`**Bandstructure**(*lattice*, *param*, *location=None*, *filename=None*)

Bases: `object`

Handles the read in, generation and storrage of the bandstructure and its relevant parameters.

Parameters

lattice [object] A *Lattice()* object.

param [object] A *Param()* object.

filename [string, optional] Filename and relative path for the input file that is to be read.

Notes

The YAML general and bandstructure configuration (param.yml and bandparams.yml, respectively by default) files are read and the setup of the bandstructure is determined from these files.

If an external band structure is supplied, i.e. from VASP (vasprun.xml is sufficient) the bandparams file is still needed as it contains parameters used to set up the scattering properties etc. at a later stage.

Presently the following combination is possible:

- Parametrized bands (parabolic, non-parabolic, Kane and a k^4 model)
- VASP input from the VASP XML file (only this file is needed)
- Numpy input files

A combination of parametrized and tight binding bands is possible.

If another code is to be used to provide the bandstructure parameters, please consult `interface` and use that as a base to write a new *bandstructure_yourcode* function in that module and include a call to this function in the initializer below.

Todo: Add the posibility to add parameterized bands to the e.g. VASP bandstructure. Usefull for instance to investigate defects etc. that was not included in the calculation.

Todo: Add interfaces to other first principle codes.

apply_scissor_operator(*self*, *energies=None*)

Apply scissor operator to blue or redshift the conduction band energies.

Parameters

energies [ndarray, optional]

Dimension: (N,M)

The energy dispersion for N bands at M k-points. Defaults to the *energies* stored in the current *Bandstructure()* object.

Returns

None

Notes

Warning: Please beware that this is a rather brutal (but usefull) operation. Make sure that the valence band maximum and conduction band minimum is fetched correctly. One way to check that this works is to plot the energies along a representative direction before and after the scissor operator have been executed.

calc_density_of_states(*self*, *return_data=False*, *num_samples=None*, *auto_scale=False*, *transport=False*, *integral_method=None*, *interpol_method=None*, *interpol_type=None*)

Calculate the density of states.

Parameters

return_data [boolean, optional] If True, return the density of states data instead of storing it in the current *Bandstructure()* object. If False, set *dos_energies*, *dos_partial* and *dos_total* in the current *Bandstructure()* object.

num_samples [integers, optional] Number of energy samples. Necessary if auto_scale is set. Otherwise the *dos_num_samples* in the parameter file is used.

auto_scale [boolean, optional] If True, auto scale the energy axis to cover the supplied band structure. Otherwise the *dos_e_min* and *dos_e_max* from the parameter file is used to set up the energy range unless *transport* is set which overrides this.

transport [bool, optional] Set to True if the density of states calculated are to be used in transport calculations (i.e. to set up the scattering). This ensures that the energy range covers the requested range of chemical potential pluss / minus 'transport_energycutband' and is then padded with zeros for the remaining values down and up to the minimum and maximum values present in the *energies* entry of the *Bandstructure()* object. Using this option make it possible to calculate the density of states on a fine grid in the relevant transport region.

integral_method [string, optional] The integration method used to calculate the DOS:

"trapz" trapeziodal integration, uses `scipy.integrate.trapz()`

"simps" simpson integration, uses `scipy.integrate.simps()`

"romb" romberb integration, uses `scipy.integrate.romb()`

"tetra" tetrahedron method, uses the linear tetrahedron method implemented in Spglib (up to version 1.7.4) by A. Togo.

If not supplied, set according to *dos_integrating_method* in the general configuration file.

Returns

dos_energies [ndarray]

Dimension: (N)

Array containing the density of states energies, where N is num_samples or set by the sampling determined in the general configuration file if *auto_scale* is set to False.

dos_total [ndarray]

Dimension: (N)

Array containing the total density of states per volume unit (units 1 / eV / AA ^ 3) at N sampled energy points, where N is determined from dos_energies.

dos_partial [ndarray]

Dimension: (M, N)

Array containing the partial (for each band index M) density of states per volume unit (1 / eV / AA ^ 3) at N sampled energy points, where N is determined from dos_energies.

See also:

`scipy.integrate.trapz`

`scipy.integrate.simps`

`scipy.integrate.romb`

`calc_effective_mass`(*self*)

Calculates the effective mass tensor. Currently the effective mass tensor is not diagonalized.

Parameters

None

Returns

None

Notes

Upon complettion the effective mass is stored in the current *Bandstructure()* object.

Also, the velocities have to be precalculated before calling this routine.

`calc_velocities`(*self*, *velocities=None*, *store=True*)

Calculate the electron group velocities from the electron energy dispersion

Parameters

velocities [ndarray, optional]

Dimension: (N, M)

Contains the group velocity along a specific direction for N bands and M k-points. If supplied the velocities of the velocities are calculated, or more specifically the inverse effective mass tensor (without prefactor)

store [boolean, optional] If True, store the calculated velocities in the active *Bandstructure()* object, else return the velocity array. Defaults to True.

Returns

velocities [ndarray, optional]

Dimension: (N, 3, M)

The velocities of the velocity. Only returned if *store* is set to False.

Notes

Does not call any interpolation routines such that the velocities can be extracted directly from the electron energy dispersion by numerical differentiation. Uses the `gradient()` from NumPy.

Overwrites any entry of exising *velocities* in *bs* and sets *gen_velocities* to False if *velocities* is not supplied.

`check_dos_energy_range`(*self*, *remin*, *remax*)

Check that the energy grid of the density of states covers the range of the supplied paramters.

Parameters

remin [float] The energy in eV for the lowest requested energy.

remax [float] The energy in eV for the highest requested energy.

Returns

within [boolean] True if the endpoints are within the energy range of the stored density of states, False ortherwise.

check_energyshifts (*self*)

Check the energy shift parameters in the parameter file for consistency.

Parameters

None

Returns

None

check_velocities (*self*, *cutoff=None*)

Check that there exists realistic values for the band velocities.

Parameters

cutoff [float, optional] Cutoff value for the test in eVAA units. Defaults to 0.01.

Returns

boolean True if values are above *cutoff*, False otherwise.

fetch_dos_energies (*self*, *e_min=None*, *e_max=None*, *num_samples=None*, *auto_scale=False*)

Set up the energy array for density of states calculations.

Parameters

e_min [float] The mininum energy in eV.

e_max [float] The maximum energy in eV.

num_samples [integer] The N number of samples between *e_min* and *e_max*.

auto_scale [boolean] If True, set the energy scale from the supplied band structure, otherwise set the scale according to *e_min* and *e_max*.

Returns

ndarray

Dimension: (N)

The N energies in eV where density of states calculations are to be performed.

fetch_energies_along_line (*self*, *kstart*, *kend*, *samplings=None*, *itype=None*, *itype_sub=None*)

Calculate the energy dispersions along specific k - points.

Parameters

kstart [ndarray]

Dimension: (3)

Direct k - point coordinate of the start point of line extraction.

kend [ndarray]

Dimension: (3)

Direct k - point coordinate of the end point of line extraction.

samplings [int, optional] The number of N samples along the line. If not specified the variable is set to the the *num_kpoints_along_line* parameter in params.yml

itype [string, optional]

Can be any of:
{“linearnd”, “interpn”, “rbf”, “wildmagic”, “skw”}

The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in the general configuration file sets this.

itype_sub [string, optional]

Can be any of:
{“nearest”, “linear”}, when *itype* is set to *interpn*.
{“multiquadric”, “inverse_multiquadric”, “gaussian”, “linear”,
“cubic”, “quintic”, “thin_plate”}, when *itype* is set to *rbf*
and when the Scipy variety is used.
{“trilinear, tricubic_exact, tricubic_bspline, akima”},
when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

Returns

energies [ndarray]

Dimension: (N, M)

The energy dispersions in eV along a line for N bands and M k - points, defined by the *num_kpoints_along_line* in the general configuration file.

kpts [ndarray]

Dimension: (N, 3)

The k - point mesh for the line extraction in cartesian coordinates.

See also:

`interpolate`

Notes

The routine `interpolate()` is used to perform the interpolation of the data along the line.

`fetch_energies_at_kpoints`(*self*, *kpoint_mesh*, *itype=None*, *itype_sub=None*)
Calculate the energy dispersions at specific k - points by interpolation.

Parameters

kpoint_mesh [ndarray]

Dimension: (N, 3)

The N k - point coordinates in cartesian coordinates.

itype [string, optional]

Can be any of:
{“linearnd”, “interpn”, “rbf”, “wildmagic”, “skw”}

The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in the general configuration file sets this.

itype_sub [string, optional]

Can be any of:

{"nearest", "linear"}, when *itype* is set to *interpn*.
{"multiquadric", "inverse_multiquadric", "gaussian", "linear",
"cubic", "quintic", "thin_plate"}, when *itype* is set to *rbf*
and when the Scipy variety is used.
{"trilinear, tricubic_exact, tricubic_bspline, akima"},
when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

Returns

energies [ndarray]

Dimension: (N, M)

The energies in eV for each of the N bands and M k - points.

See also:

`interpolate`

Notes

The routine `interpolate()` is used to perform the interpolation.

`fetch_min_max_energy`(*self*)

Returns the min and max of the energy in the current *Bandstructure()* object.

Parameters

None

Returns

emin [float] The minimum energy in eV located in the current *Bandstructure()* object.

emax [float] The maximum energy in eV located in the current *Bandstructure()* object.

`fetch_velocities_along_line`(*self*, *kstart*, *kend*, *itype=None*, *itype_sub=None*)

Calculate the velocity dispersion along a line in reciprocal space.

Parameters

kstart [ndarray, optional]

Dimension: (3)

The start k - point in cartesian coordinates.

kend [ndarray]

Dimension: (3)

The end k - point in cartesian coordinates.

itype [string, optional]

Can be any of:
{"linearnd", "interpn", "rbf", "wildmagic", "skw"}

The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in the general configuration file sets this.

itype_sub [string, optional]

Can be any of:
{“nearest”, “linear”}, when *itype* is set to *interpn*.
{“multiquadric”, “inverse_multiquadric”, “gaussian”, “linear”, “cubic”, “quintic”, “thin_plate”}, when *itype* is set to *rbf* and when the Scipy variety is used.
{“trilinear, tricubic_exact, tricubic_bspline, akima”}, when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

Returns

velocities [ndarray]

Dimension: (N, 3, M)

The group velocity in units of eVAA along a line for N bands and M k - points, defined by the *num_kpoints_along_line* in the general configuration file.

kpts [ndarray]

Dimension: (M, 3)

The kpoints where the group velocity was calculated.

See also:

interpolate

Notes

The *interpolate()* is used to perform the interpolation of the data along the line.

Warning: The factor $hbar^{-1}$ is not returned and need to be included externally.

fetch_velocities_at_kpoints(*self*, *kpoint_mesh*, *itype=None*, *itype_sub=None*)
Calculate the velocity dispersions at specific k - points.

Parameters

kpoint_mesh [ndarray]

Dimension: (N, 3)

The k - point mesh for extraction in cartesian coordinates.

itype [string, optional]

Can be any of:
{“linearnd”, “interpn”, “rbf”, “wildmagic”, “skw”}

The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in the general configuration file sets this.

itype_sub [string, optional]

Can be any of:
{“nearest”, “linear”}, when *itype* is set to *interpn*.

{“multiquadric”, “inverse_multiquadric”, “gaussian”, “linear”, “cubic”, “quintic”, “thin_plate”}, when *itype* is set to *rbf* and when the Scipy variety is used.
{“trilinear, tricubic_exact, tricubic_bspline, akima”}, when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

Returns

velocities [ndarray]

Dimension: (N, 3, M)

The group velocity in units of eVAA at the N bands for M k - points.

See also:

`interpolate`

Notes

The `interpolate()` is used to perform the interpolation.

Warning: The factor $hbar^{-1}$ is not returned and need to be included externally.

gen_analytic_band(*self*, *band*)
Generate an analytical energy and velocity dispersion.

Parameters

band [int] The band index used to fetch band parameters in bandparams.yml.

Returns

energy [ndarray]

Dimension: (M)

The energy dispersion in eV at M k-points.

velocity [ndarray]

Dimension: (M,3)

The group velocity in eVAA of the energy dispersion (without the $hbar^{-1}$ factor).

gen_bands(*self*)
Generates the set of energy and velocity dispersions.

Parameters

None

Returns

energies [ndarray]

Dimension: (N,M)

Contains the energy dispersions in eV for N bands at M k-points.

velocities [ndarray]

Dimension: (N,M,3)

Contains the group velocities in eVAA of the energy dispersions (without the $hbar^{-1}$ factor).

tb_band [ndarray]

Dimension: (N)

Contains boolean values of True for band indexes that are tight binding bands, False otherwise.

gen_dos (*self*)

Generates the density of states for the analytic models

Parameters

None

Returns

dos [ndarray]

Dimension: (N, M)

Contains the density of states for N bands at M *dos_num_samples* from *dos_e_min* to *dos_e_max*. The number of samplings and their range is set it general configuration file. Units are per volume unit, 1/eV/AA^3.

dos_energies [ndarray]

Dimension: (M)

Contains the energy samplings of which *dos* was calculated in units of eV.

Notes

Currently only the parabolic models are implemented.

Todo: Also implement the non-parabolic alpha models

interpolate (*self*, *iksampling=None*, *ienergies=True*, *ivelocities=False*, *itype=None*, *itype_sub=None*, *kpoint_mesh=None*, *store_inter=False*, *energies=None*)

Interpolates the energies and velocity dispersion.

Parameters

iksampling [ndarray, optional]

Dimension: (3)

Contains the interpolated k - point mesh sampling values. Does not have to be set if line extraction performed (full grid is instead supplied in line_mesh).

ienergies [boolean] If True, interpolate the energies, if not, do not.

ivelocities [boolean] If True, interpolate the velocities, if not, do not

itype [string, optional] Can be any of: {"linearnd", "interpn", "rbf", "wildmagic", "skw"} The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in the general configuration file sets this.

itype_sub [string, optional]

Can be any of:
{"nearest", "linear"}, when *itype* is set to *interpn*.
{"multiquadric", "inverse_multiquadric", "gaussian", "linear",
"cubic", "quintic", "thin_plate"}, when *itype* is set to *rbf*
and when the Scipy variety is used.
{"trilinear, tricubic_exact, tricubic_bspline, akima"},
when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

kpoint_mesh [ndarray, optional]

Dimension: (M, 3)

Supplied k - point grid for extraction as an alternative to iksampling. Should be supplied in cartesian coordinates and should not extend the border of the original grid. Usefull for line extraction etc.

store_inter [boolean, optional] Store the new interpolated energies and velocities in the supplied object. Also modifies the current *Lattice()* object with the new grid etc. if that has been modified. Defaults to False.

energies [ndarray, optional]

Dimension: (N,J)

An input array containing the energies (or some other) quantity that can fly through with the same structure, e.g. the velocities along a certain direction for N bands and J k-points.

Returns

ien, ivel [ndarray, ndarray]

Dimension: (N,M), (N,3,M)

The energy dispersions in eV, and group velocities in eVAA along indexes each axis of the reciprocal basis are returned for N bands and M new k - points if velocities is supplied, or if the *gen_velocities* tag is set to True in the current *Bandstructure()* object.

ien, False, False, False [ndarray, boolean, boolean, boolean]

Dimension: (N,M)

The energy dispersions in eV for N bands and M new k-points if *velocities* is not supplied or *gen_velocities* is set to False.

See also:

`linearnd`

`interpn`

`rbf`

Notes

Todo: DOCUMENT THE DIFFERENT INTERPOLATION SCHEMES, OR AT LEAST ADD PROPER REFERENCES.

`locate_band_gap`(*self*)
Calculate the band gap.

Parameters

None

Returns

float The band gap in eV

`locate_bandgap`(*self*, *energies=None*, *occ=None*)
Locate the band gap.

Parameters

energies [ndarray, optional]

Dimension: (N,M)

The energy dispersion in eV for N bands at M k-points. Defaults to the *energies* stored in the current *Bandstructure()* object.

occ [ndarray, optional]

Dimension: (N,M)

The occupancy for N bands at M k-points. Defaults to the *occ* stored in the current *Bandstructure()* object.

Returns

vbm_energy [float] The valence band maximum in eV.

bandgap [float] The band gap in eV.

`locate_cbm`(*self*, *energies*, *occ*)
Locate the conduction band minimum.

Parameters

energies [ndarray]

Dimension: (N,M)

The energy dispersion in eV for N bands at M k-points.

occ [ndarray]

Dimension: (N,M)

The occupancy for N bands at M k-points.

Returns

energy [float] The conduction band minimum in eV.

band [int] The band index of the conduction band minimum.

kpoint [int] The kpoint index of the conduction band minimum.

locate_vbm(*self*, *energies*, *occ*)
Locate the valence band maximum.

Parameters

energies [ndarray]

Dimension: (N,M)

The energy dispersion in eV for N bands at M k-points.

occ [ndarray]

Dimension: (N,M)

The occupancy for N bands at M k-points.

Returns

energy [float] The valence band maximum in eV

band [int] The band index of the valence band maximum.

kpoint [int] The kpoint index of the valence band maximum.

bandstructure.**gaussian**(*energy*, *energy_ref*, *smearing*)
Returns the value of a Gaussian function.

Parameters

energy [float] The energy in eV.

energy_ref [float] The reference energy in eV.

smearing [float] The smearing factor in eV.

Returns

float The value in eV.

bandstructure.**non_parabolic_energy_1**(*k*, *effmass*, *a*, *scale*, *e0=0.0*, *kshift=None*)
Calculates a energy dispersion, both parabolic and non-parabolic.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point coordinates (cartesian) where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^2k^4 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

e0 [float, optional] Shift of the energy scale in eV.

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

ndarray

Dimension: (N)

Contains the energy dispersion in eV at each N k-points.

Notes

This routines calculates the energy dispersion according to

$$E =$$
$$frac$$
$$hbar^2 k^2 2m + ak^4.$$

Setting a to zero yields a parabolic dispersion.

`bandstructure.`**`non_parabolic_energy_2`**(*k*, *effmass*, *a*)

Calculates a non-parabolic energy dispersion.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point cartesian coordinates where the dispersion is to be evaluated.

effmass [float] The effective mass in units of the free electron mass.

a [float] The
$alpha$ factor.

Returns

ndarray

Dimension: (N)

Contains the energy in eV at each N k-points for each direction defined by the direction of the k-point unit axis.

Notes

This routine calculates the energy dispersion according to

$$E(1+$$
$$alphaE) =$$
$$frac$$
$$hbar^2 k^2 2m,$$

where
$alpha$ is a parameter that adjust the non-parabolicity

Note that if m is negative (valence bands), the square root is undefined for $k^2 >= m/(2$
$alpha$

$hbar)$, which is a rather limited k-space volume. Consider (either m or $alpha$ negative).

$$m = m_e,\\ alpha = 1.0(E_g = 1.0\\ mathrmeV)\\ rightarrow|\\ veck|\\ geq 0.26\\ mathrmAA^{-1}$$

$$m = 0.1m_e,\\ alpha = 1.0\\ rightarrow|\\ veck|\\ geq 0.081\\ mathrmAA^{-1}$$

$$m = 10m_e,\\ alpha = 1.0\\ rightarrow|\\ veck|\\ geq 0.81\\ mathrmAA^{-1}$$

$$m = m_e,\\ alpha = 10.0(E_g = 0.1\\ mathrmeV)\\ rightarrow|\\ veck|\\ geq 0.81\\ mathrmAA^{-1}$$

$$m = m_e,\\ alpha = 0.1(E_g = 10\\ mathrmeV)\\ rightarrow|\\ veck|\\ geq 0.081\\ mathrmAA^{-1}$$

For a simple cell of 10
$mathrmAA$, the BZ border is typically at 0.31
$mathrmAA^{-1}$ and for a smaller cell, e.g. 3.1
$mathrmAA$, the BZ border is here at 1.0
$mathrmAA^{-1}$.

Warning: In order to be able to use all values of a, we return a linear $E($
$veck)$ in the undefined region (the last defined value of $E($
$veck)$ is used). This is highly unphysical, so we print a warning to notice the user

`bandstructure.`**`non_parabolic_energy_3`**(*k*, *effmass*, *a*, *scale*, *e0=0.0*, *kshift=None*)

Calculates a k^2 + k^6 energy dispersion.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point coordinates (cartesian) where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^4k^8 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

e0 [float, optional] Shift of the energy scale in eV.

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

ndarray

Dimension: (N)

Contains the energy dispersion in eV at each N k-points.

Notes

This routines calculates the energy dispersion according to

$$E =$$
$$frac$$
$$hbar^2k^22m + a^3k^6.$$

Setting a to zero yields a parabolic dispersion.

`bandstructure.`**`non_parabolic_energy_4`**(*k*, *effmass*, *a*, *scale*, *e0=0.0*, *kshift=None*)

Calculates a k^2 + k^8 energy dispersion.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point coordinates (cartesian) where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^4k^8 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

e0 [float, optional] Shift of the energy scale in eV.

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

ndarray

Dimension: (N)

Contains the energy dispersion in eV at each N k-points.

Notes

This routines calculates the energy dispersion according to

$$E =$$
$$frac$$
$$hbar^2k^22m + a^4k^8.$$

Setting a to zero yields a parabolic dispersion.

`bandstructure.`**`non_parabolic_energy_5`**(*k*, *_*, *a*, *scale*, *e0=0.0*, *kshift=None*)

Calculates a linear energy dispersion.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point coordinates (cartesian) where the dispersion is to be evaluated.

_ [A dummy.]

a [ndarray]

Dimension: (3)

The coefficients in front of each k^2 direction which translates to $sqrt(a)k$ in the one dimensional case.

scale [float] The scale factor in front of the linear expression.

e0 [float, optional] Shift of the energy scale in eV.

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

ndarray

Dimension: (N)

Contains the energy dispersion in eV at each N k-points.

Notes

This routines calculates the energy dispersion according to (in one dimension)

$$E = a$$
$$sqrt(k^2).$$

Multiplied by the scale factor in front of this.

`bandstructure.`**`non_parabolic_velocity_1`**(*k*, *effmass*, *a*, *scale*, *kshift=None*)

Calculates the group velocity for the energy dispersion generated in `non_parabolic_energy_1()`.

For both parabolic and non-parabolic bands.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point in cartesian coordinates where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^2k^4 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

vx, vy, vz [ndarray, ndarray, ndarray]

Dimension: (N),(N),(N)

Contains the group velocity at each N k-points for each direction defined by the direction of the k-point unit axis. Units of eVAA.

Notes

This routines calculates the group velocity according to

$$v =$$
$$frac$$
$$partial E$$
$$partial$$
$$veck,$$

where

$$E =$$
$$frac$$
$$hbar^2 k^2 2m + ak^4.$$

Setting a to zero yields a parabolic dispersion and thus its group velocity.

Warning: The factor
$hbar^{-1}$ is not returned and need to be included externally.

`bandstructure.`**`non_parabolic_velocity_2`**`(`*k*, *effmass*, *a*`)`

Calculates the group velocity for the energy dispersion generated in *`non_parabolic_energy_2()`*.

For both parabolic and non-parabolic.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point in cartesian coordinates where the dispersion is to be evaluated.

effmass [float] The effective mass in units of the free electron mass.

a [ndarray]

Dimension: (3)

The
$alpha$ factor.

Returns

vx, vy, vz [ndarray, ndarray, ndarray]

Dimension: (N), (N), (N)

The group velocity along each axis in the reciprocal unit cell. In units of eVAA.

Notes

Consult comments in `non_parabolic_energy_1()`

> **Warning:** The factor
> $hbar^{-1}$ is not returned and need to be included externally.

`bandstructure.`**`non_parabolic_velocity_3`**(*k*, *effmass*, *a*, *scale*, *kshift=None*)

Calculates the group velocity for the energy dispersion generated in `non_parabolic_energy_3()`.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point in cartesian coordinates where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^4k^8 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

vx, vy, vz [ndarray, ndarray, ndarray]

Dimension: (N),(N),(N)

Contains the group velocity at each N k-points for each direction defined by the direction of the k-point unit axis. Units of eVAA.

Notes

This routines calculates the group velocity according to

$$v =$$
$$frac$$
$$partialE$$
$$partial$$
$$veck,$$

where

$$E = \\ frac \\ hbar^2k^22m + a^3k^6.$$

Setting a to zero yields a parabolic dispersion and thus its group velocity.

Warning: The factor
$hbar^{-1}$ is not returned and need to be included externally.

`bandstructure.`**`non_parabolic_velocity_4`**(*k*, *effmass*, *a*, *scale*, *kshift=None*)

Calculates the group velocity for the energy dispersion generated in *`non_parabolic_energy_4()`*.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point in cartesian coordinates where the dispersion is to be evaluated.

effmass [ndarray]

Dimension: (3)

Contains the effective mass along the three k-point directions. Only the diagonal components of the effective mass tensor is used. In units of the free electron mass.

a [ndarray]

Dimension: (3)

The non parabolic coefficients in front of each k^2 direction which translates to a^4k^8 in the one dimensional case.

scale [float] The scale factor in front of the non-parabolic correction

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

vx, vy, vz [ndarray, ndarray, ndarray]

Dimension: (N),(N),(N)

Contains the group velocity at each N k-points for each direction defined by the direction of the k-point unit axis. Units of eVAA.

Notes

This routines calculates the group velocity according to

$$v = \\ frac \\ partial E \\ partial \\ vec k,$$

where

$$E = \\ frac \\ hbar^2 k^2 2m + a^4 k^8.$$

Setting a to zero yields a parabolic dispersion and thus its group velocity.

Warning: The factor
$hbar^{-1}$ is not returned and need to be included externally.

`bandstructure.`**`non_parabolic_velocity_5`**(*k*, *_*, *a*, *scale*, *kshift=None*)

Calculates the group velocity for the energy dispersion generated in *`non_parabolic_energy_5()`*.

Parameters

k [ndarray]

Dimension: (N,3)

Contains the N k-point in cartesian coordinates where the dispersion is to be evaluated.

_ [A dummy.]

a [ndarray]

Dimension: (3)

The coefficients in front of each k^2 direction which translates to
$sqrt(a)k$ in the one dimensional case.

scale [float] The scale factor in front of the linear expression.

kshift [ndarray, optional]

Dimension: (3)

The shift along the respective k-point vectors in cartesian coordinates.

Returns

vx, vy, vz [ndarray, ndarray, ndarray]

Dimension: (N),(N),(N)

Contains the group velocity at each N k-points for each direction defined by the direction of the k-point unit axis. Units of eVAA.

Notes

This routines calculates the group velocity according to

$$v =$$
$$frac$$
$$partial E$$
$$partial$$
$$veck,$$

where

$$E = a$$
$$sqrt(k^2).$$

Multiplied by the scale factor in front of this.

> **Warning:** The factor
> $hbar^{-1}$ is not returned and need to be included externally.

`bandstructure.`**`parabolic_effective_mass`**(*effmass_t*)
Checks if the supplied effective mass array is parabolic.

Parameters

effmass_t [ndarray] The effective mass tensor in units of the free electron mass.

Returns

boolean True if parabolic tensors, False otherwise.

6.10.8 inputoutput module

This module, `inputoutput.py` handles input and output related acitivites.

Contains various input and output routines for T4ME.

`class` `inputoutput.`**`Param`**(*data*)
Bases: `object`

YAML reader for the input paramters.

Parameters

data [iterable] yaml load (typically safe_load(open(yamlfilename),"r")).

Notes

Read a YAML paramter file.

`inputoutput.`**`dump_bandstruct_line`**(*bs, kstart, kend, filename='band', datatype='e', k_direct=True, itype=None, itype_sub=None*)
Writes the energy or velocity dispersions extracted along a line to a file.

Parameters

bs [object] A *Band()* object containing the energies and velocity dispersions.

kstart [ndarray]

Dimension: (3)

The start k-point vector in cartesian coordinates.

kend [ndarray]

Dimension: (3)

The end k-point vector in cartesian coordinates.

filename [string, optional] The filename used to write the energy or velocity dispersions. Defaults to "band".

datatype [{"e","v"}] Selects to write energy dispersions ("e") or velocity dispersions ("v").

itype [string, optional]

Can be any of:
{"linearnd", "interpn", "rbf", "wildmagic", "skw"}

The type of interpolate method to use. If not set, the parameter *dispersion_interpolate_method* in param.yml sets this.

itype_sub [string, optional]

Can be any of:
{"nearest", "linear"}, when *itype* is set to *interpn*.
{"multiquadric", "inverse_multiquadric", "gaussian", "linear",
"cubic", "quintic", "thin_plate"}, when *itype* is set to *rbf*
and when the Scipy variety is used.
{"trilinear, tricubic_exact, tricubic_bspline, akima"},
when *itype* is set to *wildmagic*.

The subtype of the interpolation method.

Returns

None

`inputoutput.`**`dump_density_of_states`**(*bs*, *dos=None*, *dos_energies=None*, *filename='dos'*)
Writes the density of states to file.

Parameters

bs [object] A *Bandstructure()* object.

dos [ndarray, optional]

Dimension: (N,M)

The density of states for N bands at M energy samplings If not supplied, set to *bs.dos*.

dos_energies [ndarray, optional]

Dimension: (M)

The M energy samples used for the density of state If not supplied, set to *bs.dos_energies*

filename [string, optional] The filename used to write the density of states. Default is "dos".

`inputoutput.`**`dump_relaxation_time`**(*tr*, *filename=None*)
Writes the relaxation time to file.

Parameters

tr [object] A *Transport()* object containing the relaxation time and other details related to the carrier transport.

filename [string, optional] The output filename, default is "scattering". The string "_band_N" is added to this string, where N is the band number.

Returns

None

Notes

One file per band, filename "scattering_band_x", where x is the band number. In each file the temperature dependence is blocked, while the carrier energy, total relaxation time and each individual relaxation times follow as columns for each block.

`inputoutput.`**`dump_transport_coefficients`**(*tr*, *filename_tag=None*)

Writes the transport coefficients to files

Parameters

tr [object] A *Transport()* object that contains the transport coefficients

filename_tag [string, optional]

If *filename_tag* is not an empty string, but a string *x*, the
output filenames are:
sigma_x: contains the electrical conductivity in units of

mathrmS/
mathrmm
seebeck_x: contains the Seebeck coefficiens in units of

mu
mathrmV/
mathrmK
lorenz_x: contains the Lorenz number in units of
10^{-8}
mathrmV^2/
mathrmK^2
kappa_x: contains the Seebeck coefficiens in units of

mu
mathrmW/
mathrmmK
hall_x: the Hall coefficient (big R) in units of

mathrmcm^3/
mathrmC
cc_x: the carrier concentration in units of
10^{21}
mathrmcm^{-3}

The default is to write the files without the tag on the end. Consult header of the files for the ordering.

Returns

None

Notes

Each temperature steps have its own block.

`inputoutput.`**`end_message`**`()`

Prints an end message to the log file.

Parameters

None

Returns

None

`inputoutput.`**`file_handler`**`(`*filename=”*, *handler=None*, *status=None*`)`

Open and close files

Parameters

filename [string] Filename to be handled

handler [object, optional] A file object. If provided, this routine closes the file

status [{“w”, “r”, “a”}] The status, e.g. write, read, append etc.

Returns

file_handler [object] A file object

`inputoutput.`**`readbandparam`**`(`*location=None*, *filename=None*`)`

Load the parameters in the bandstructure configuration file.

Parameters

location [string, optional] The location of the bandstructure configuration file. Defaults to “input” directory in the current working directory.

filename [string, optional] The filename for the bandstructure configuration file. Defaults to “bandparam.yml”.

Returns

iterable An iterable YAML object.

Notes

The current working directory is padded in front of any supplied location (or if path is given in the filename).

`inputoutput.`**`readcellparam`**`(`*location=None*, *filename=None*`)`

Load the parameters in the cell configuration file.

Parameters

location [string, optional] The location of the cell configuration file. Defaults to the “input” directory in the current working directory.

filename [string, optional] The filename for the cell configuration file. Defaults to “cellparam.yml”.

Returns

iterable An iterable YAML object.

Notes

The current working directory is padded in front of any supplied location (or if path is given in the filename).

`inputoutput.`**`readparam`**`(`*location=None*, *filename=None*`)`
Load the parameters in the general configuration file.

Parameters

location [string, optional] The location of the general configuration file. Defaults to the "input" directory in the current working directory.

filename [string, optional] The filename for the general configuration file. Defaults to "param.yml".

Returns

iterable An iterable YAML object.

Notes

The current working directory is padded in front of any supplied location (or if path is given in the filename).

`inputoutput.`**`skw_warning`**`()`
An error for missing SKW.

`inputoutput.`**`spglib_error`**`()`
An error for missing Spglib interface.

`inputoutput.`**`start_message`**`()`
Prints a startup message to the log file.

Parameters

None

Returns

None

`inputoutput.`**`wildmagic_warning`**`()`
An error for a missing GeometricTools interface.

6.10.9 interface module

This module, `interface.py` contains the interfaces which reads in data and calculates for instance parametrized bandstructures etc.

Contains routines that interface T4ME, e.g. to the parameter files or to other input files.

`interface.`**`bandstructure_numpy`**`(`*bs*, *filename*, *location=None*`)`
Sets the bandstructure from a NumPy datafile file.

Loads and stores the parameters in the bandstructure configuration file (defaults to bandparam.yml).

Parameters

bs [object] A *Bandstructure()* object.

filename [string] The filename of the NumPy data file to be read. The bandstructure configuration file have to be named "bandparam.yml" in this case.

location [string, optional] The location of the NumPy data file. Defaults to the "input" directory in the current working directory.

Returns

None

Notes

This routine read NumPy datafiles containing the electron energy dispersions and optionally the band velocities.

The datastructure of the supplied numpy array
should be on the following format:
[
[kx], [ky], [kz], [e_1], [v_x_1], [v_y_1], [v_z_1],
[e_2], [v_x_2], [v_y_2], [v_z_2], ... ,
[e_n], [v_x_n], [v_y_n], [v_z_n]
]

If the band velocities are not supplied they are simply not present. Each column of data has the length of the number of k-point in the full BZ.

The bandstructure configuration file is still read due to the need of the scattering properties etc.

This interface is enabled by setting *read* in the general configuration file to "numpy" (datafile with only electron energy dispersions) or "numpyv" (datafile with electron energy and group velocity dispersion)

`interface.`**`bandstructure_param`**(*bs*, *location=None*, *filename=None*)

Sets the bandstructure from the parameters in the bandstructure configuration file (default bandparam.yml).

Also loads and stores the parameters.

Parameters

bs [object] A *Bandstructure()* object.

location [string, optional] The location of the bandstructure configuration file. Defaults later to the "input" directory in the current working directory.

filename [string, optional] The filename of the bandstructure configuration file. Defaults to "bandparam.yml".

Returns

None

Notes

This interface prepares analytic and tight binding generation of the band structure and also loads and stores all bandstructure related parameters.

`interface.`**`bandstructure_vasp`**(*bs*, *location=None*, *filename=None*)
Sets the bandstructure from a VASP XML file.

Loads and stores the parameters in the bandstructure configuration file (defaults to bandparam.yml).

Parameters

bs [object] A *Bandstructure()* object.

location [string, optional] The location of the VASP XML file. Defaults to the “input” directory in the current working directory.

filename [string, optional] The filename of the VASP XML file to be read. Defaults to “vasprun.xml”. The bandstructure configuration file have to be named “bandparam.yml” in this case.

Returns

None

Notes

This interface read and sets up the bandstructure based on a VASP XML file. Currently it does not read the band velocities as VASP does not yet support this feature. However, work is in progress to enable this. The band velocities have to be generated by an interpolation routine later. Flags are automatically set for this. The bandstructure configuration file is still read due to the need of the scattering properties etc.

This interface is enabled by setting *read* in the general configuration file to vasp.

`interface.`**`lattice_param_numpy`**(*lattice*, *location=None*, *filename=None*)
Interface used to format the elements needed to generate the *Lattice()* object.

Used if the lattice is generated from the celldata YAML file (parameterfile and Numpy intput files).

Parameters

lattice [object] A *Lattice()* object where we can store additional parameters detected during setup for later access.

location [string, optional] The location of the YAML parameter file determining the celldata.

filename [string, optional] The filename of the YAML parameter file determining the celldata.

Returns

unitcell [ndarray]

Dimension: (3,3)

The unitcell in cartesian coordinates and {AA} units.

positions [ndarray]

Dimension: (N,3)

The positions of the N atoms in the unitcell in cartesian coordinates.

species [ndarray]

Dimension: (N)

Integer atomic numbers of the atomic species in the same order as positions. Hydrogen starts with 1, while the element X is located at 0. Otherwise it follows the periodic table.

kmesh [object]

Dimension: (3)

A *Kmesh()* obhect for the reciprocal mesh generation containment. Should include *sampling*, *mesh*, *mesh_ired* and other parameters needed for later processing.

Notes

Upon writing a custom interface, please make sure that the parameters in the YAML files are not overwritten.

`interface.`**`lattice_vasp`**(*lattice*, *location=None*, *filename=None*)

Interface used to format the elements needed to generate the *Lattice()* object.

Used if the lattice is generated from the VASP XML file.

Parameters

lattice [object] A *Lattice()* object where we can store additional parameters detected during setup for later access.

location [string, optional] The location of the VASP XML file determining the celldata.

filename [string, optional] The filename of the VASP XML file determining the celldata.

Returns

unitcell [ndarray]

Dimension: (3,3)

The unitcell in cartesian coordinates and {AA} units.

positions [ndarray]

Dimension: (N,3)

The positions of the N atoms in the unitcell in cartesian coordinates.

species [ndarray]

Dimension: (N)

Integer atomic numbers of the atomic species in the same order as positions. Hydrogen starts with 1, while the element X is located at 0. Otherwise it follows the periodic table.

kmesh [object]

Dimension: (3)

A *Kmesh()* object for the reciprocal mesh generation containment. Should include *sampling*, *mesh*, *mesh_ired* and other parameters needed for later processing.

Notes

Upon writing a custom interface, please make sure that the parameters in the YAML parameter files are not overwritten.

Additional parameters pertaining VASP are stored inside the *Param()* object with a *vasp* preemble, i.e. *param.vasp_something*.

`interface.`**`lattice_w90`**(*lattice*)

Interface used to format the elements needed to generate the *Lattice()* object.

Used if the lattice is generated from the Wannier90 win file.

Parameters

lattice [object] A *Lattice()* object where we can store additional parameters detected during setup for later access.

Returns

unitcell [ndarray]

Dimension: (3,3)

The unitcell in cartesian coordinates and {AA} units.

positions [ndarray]

Dimension: (N,3)

The positions of the N atoms in the unitcell in cartesian coordinates.

species [ndarray]

Dimension: (N)

Integer atomic numbers of the atomic species in the same order as positions. Hydrogen starts with 1, while the element X is located at 0. Otherwise it follows the periodic table.

kmesh [object] A *Kmesh()* object for the reciprocal mesh generation containment. Should include *sampling*, *mesh*, *mesh_ired* and other parameters needed for later processing.

Notes

Upon writing a custom interface, please make sure that the parameters in the YAML parameter file is not overwritten.

Additional parameters pertaining VASP are stored inside the *Param()* object with a *vasp* preemble, i.e. *param.vasp_something*.

`interface.`**`read_band_parameters`**(*bs*, *numbands*, *location=None*, *filename=None*)

Reads and stores the information in the band parameters configuration file (bandparam.yml).

Parameters

bs [object] The active *Bandstructure()* object.

numbands [int] The number of bands

location [string, optional] The folder in which the band configuration file is placed. Defaults to the relative folder “input”.

filename [string, optional] The filename of the band configuration file. Defaults to bandparam.yml.

Returns

None

Notes

Reads and stores the values in the band configuration file. When writing custom interfaces it is sufficient to call this routine in order for the setup of the individual band parameters to be consistent.

6.10.10 utils module

This module, `utils.py` contains general routines.

Containing utilitary functions for T4ME.

`utils.`**`check_directory`**(*path*, *create=False*)

Check that a directory exists.

Parameters

path [string] The path to the directory to be checked.

create [boolean, optional] If set to True create the directory if it does not exist. Defaults to False and in this case an error is printed if the directory is not found.

Returns

None

`utils.`**`check_file`**(*filename*)

Check if a file exists

Parameters

filename [string] The location and filename of the file to be checked.

Returns

None

`utils.`**`clean_directory`**(*path*)

Clean a directory.

Parameters

path [string] The path to the directory to be cleaned.

Returns

None

`utils.`**`config_logger`**(*filename='/logging.yaml'*, *level=None*)

Configure the main logger.

Parameters

filename [string, optional] The filename for the logging configuration file. Defaults to "logging.yaml" in the current working directory.

level [object] Sets the logging level of the Python logger. Defaults to *INFO* if the configuration file is not found.

Returns

None

`utils.`**`create_directory`**(*path*)

Check that the directory exists

Parameters

path [string] The path to the directory to be checked.

Returns

None

`utils.`**`fetch_sorting_indexes`**`(`*data*, *order='C'*`)`

Fetch the sorting indexes to sort an array to either column or row order.

Parameters

data [ndarray]

Dimension: (N,M)

The input data array to be sorted.

order [{"C", "F"}] The sort order.

Returns

sort_index [ndarray]

Dimension: (N,M)

The sorting indexes that can be used to order the array.

`utils.`**`invert_matrix`**`(`*matrix*`)`

Inverts a matrix and checks for ill-conditions

Parameters

matrix [ndarray]

Dimension: (N,N)

The input matrix to be inverted.

Returns

inv_matrix [ndarray]

Dimension: (N,N)

The inverted matrix. If *matrix* is ill-conditioned, nan values are filled in the matrix.

`utils.`**`is_even`**`(`*number*`)`

Check if *number* is even.

Parameters

number [integer] The integer to be checked

Returns

boolean Returns True of *number* is even, False otherwise.

`utils.`**`is_number`**`(`*something*`)`

Check if *something* is a number.

Parameters

something [anything] Something to be checked.

Returns

boolean True if *something* is a number. False otherwise.

`utils.`**`is_power_of_two`**`(`*number*`)`

Check that if a number is a power of two.

Parameters

number [float] The supplied number to be checked.

Returns

boolean Returns True of *number* is power of two, False otherwise.

utils.**pull_points_back_into_zone**(*points*)
Pulls all points outside [-0.5,0.5] in direct coordinates back into [-0.5, 0.5].

Parameters

points [ndarray]

Dimension: (N, 3)

The N points to be checked and thrown back into the zone (between [-0.5, 0.5]). Should be in direct coordinates.

Returns

None

utils.**pull_vecs_inside_boundary**(*vecs*, *border*, *shift=None*)
Pulls vectors into a given cubic boundary box.

Parameters

vecs [ndarray]

Dimension: (N,3)

Contains N vectors.

border [ndarray]

Dimension: (6)

Contains the entries x_min, x_max, y_min, y_max, z_min and z_max, respectively of the indexes to be modified, typically the border elements.

shift [float, optional] An optional shift value which brings the vectors shift more inside the boundary box supplied in *border*.

Returns

None

Python Module Index

b

bandstructure, 87

i

inputoutput, 109
interface, 113

l

lattice, 81
lbtecoeff, 65
lbteint, 59

s

scattering, 48

t

t4me, 54
transport, 54

u

utils, 118

Index

A

acceptor_ionization() (*in module transport*), 57
analytic_k_space_energy() (*in module lbteint*), 59
analytic_k_space_integrand() (*in module lbteint*), 59
analytic_k_space_velocity() (*in module lbteint*), 59
apply_scissor_operator() (*bandstructure.Bandstructure method*), 88

B

Bandstructure (*class in bandstructure*), 87
bandstructure (*module*), 87
bandstructure_numpy() (*in module interface*), 113
bandstructure_param() (*in module interface*), 114
bandstructure_vasp() (*in module interface*), 114

C

calc_carrier_concentration() (*transport.Transport method*), 54
calc_density_of_states() (*bandstructure.Bandstructure method*), 89
calc_effective_mass() (*bandstructure.Bandstructure method*), 90
calc_transport_tensors() (*transport.Transport method*), 54
calc_velocities() (*bandstructure.Bandstructure method*), 90
calculate_cell_volume() (*in module lattice*), 87
calculate_hall_carrier_concentration() (*in module lbtecoeff*), 65
calculate_hall_factor() (*in module lbtecoeff*), 65
cart_to_dir() (*lattice.Lattice method*), 81
check_directory() (*in module utils*), 118
check_dos_energy_range() (*bandstructure.Bandstructure method*), 90
check_energyshifts() (*bandstructure.Bandstructure method*), 91
check_file() (*in module utils*), 118
check_for_duplicate_points() (*lattice.Lattice method*), 82
check_lattice() (*lattice.Lattice method*), 82
check_mesh() (*lattice.Lattice method*), 82
check_scattering() (*in module scattering*), 48
check_sensible_ksampling() (*in module lattice*), 87
check_velocities() (*bandstructure.Bandstructure method*), 91
clean_directory() (*in module utils*), 118
combined_scattering() (*in module scattering*), 49
concatenate_integrand() (*in module lbteint*), 60
concatenate_integrand_band() (*in module lbteint*), 60
config_logger() (*in module utils*), 118
create_directory() (*in module utils*), 118
create_kmesh() (*lattice.Lattice method*), 82

D

dir_to_cart() (*lattice.Lattice method*), 83
donor_ionization() (*in module transport*), 58
dump_bandstruct_line() (*in module inputoutput*), 109
dump_density_of_states() (*in module inputoutput*), 110
dump_relaxation_time() (*in module inputoutput*), 110
dump_transport_coefficients() (*in module inputoutput*), 111

E

end_message() (*in module inputoutput*), 112

F

fermi_dist() (*in module transport*), 58
fermiintclosed() (*in module lbteint*), 60
fetch_bz_border() (*lattice.Lattice method*), 84
fetch_chempot_from_etas() (*in module transport*), 58
fetch_chempots() (*transport.Transport method*), 56
fetch_dos_energies() (*bandstructure.Bandstructure method*), 91
fetch_energies_along_line() (*bandstructure.Bandstructure method*), 91
fetch_energies_at_kpoints() (*bandstructure.Bandstructure method*), 92
fetch_etas() (*transport.Transport method*), 56
fetch_iksampling() (*lattice.Lattice method*), 84
fetch_kmesh() (*lattice.Lattice method*), 84
fetch_kmesh_step_size() (*lattice.Lattice method*), 84
fetch_kmesh_unit_vecs() (*lattice.Lattice method*), 85
fetch_kpoints_along_line() (*lattice.Lattice method*), 85
fetch_ksampling_from_stepsize() (*lattice.Lattice method*), 85
fetch_length_runitcell_vecs() (*lattice.Lattice method*), 86
fetch_min_max_energy() (*bandstructure.Bandstructure method*), 93
fetch_relevant_bands() (*transport.Transport method*), 56
fetch_sorting_indexes() (*in module utils*), 118
fetch_temperatures() (*transport.Transport method*), 57
fetch_velocities_along_line() (*bandstructure.Bandstructure method*), 93
fetch_velocities_at_kpoints() (*bandstructure.Bandstructure method*), 94
file_handler() (*in module inputoutput*), 112
find_r_for_closed() (*in module scattering*), 49

G

gaussian() (*in module bandstructure*), 99
gen_analytic_band() (*bandstructure.Bandstructure method*), 95
gen_bands() (*bandstructure.Bandstructure method*), 95
gen_dos() (*bandstructure.Bandstructure method*), 96
generate_consistent_mesh() (*lattice.Lattice method*), 86

I

inputoutput (*module*), 109
integrandpar() (*in module lbteint*), 60
integrandpardos() (*in module lbteint*), 61
integrandpart2() (*in module lbteint*), 62
interface (*module*), 113
interpolate() (*bandstructure.Bandstructure method*), 96
interpolate() (*in module scattering*), 50
invert_matrix() (*in module utils*), 119
is_even() (*in module utils*), 119
is_number() (*in module utils*), 119
is_power_of_two() (*in module utils*), 119

L

Lattice (*class in lattice*), 81
lattice (*module*), 81
Lattice.Kmesh (*class in lattice*), 81
lattice_param_numpy() (*in module interface*), 115
lattice_vasp() (*in module interface*), 116
lattice_w90() (*in module interface*), 116
lbtecoeff (*module*), 65
lbteint (*module*), 59
locate_band_gap() (*bandstructure.Bandstructure method*), 98
locate_bandgap() (*bandstructure.Bandstructure method*), 98
locate_cbm() (*bandstructure.Bandstructure method*), 98
locate_vbm() (*bandstructure.Bandstructure method*), 98

N

non_parabolic_energy_1() (*in module bandstructure*), 99
non_parabolic_energy_2() (*in module bandstructure*), 100
non_parabolic_energy_3() (*in module bandstructure*), 102
non_parabolic_energy_4() (*in module bandstructure*), 102
non_parabolic_energy_5() (*in module bandstructure*), 103
non_parabolic_velocity_1() (*in module bandstructure*), 104
non_parabolic_velocity_2() (*in module bandstructure*), 105
non_parabolic_velocity_3() (*in module bandstructure*), 106
non_parabolic_velocity_4() (*in module bandstructure*), 107
non_parabolic_velocity_5() (*in module bandstructure*), 108
numerick() (*in module lbtecoeff*), 66

P

pad_scattering_values() (*in module scattering*), 50
parabolic_closed() (*in module lbtecoeff*), 67
parabolic_effective_mass() (*in module bandstructure*), 109
parabolic_numeric() (*in module lbtecoeff*), 69
parabolice() (*in module lbtecoeff*), 80
Param (*class in inputoutput*), 109
pull_points_back_into_zone() (*in module utils*), 120
pull_vecs_inside_boundary() (*in module utils*), 120

R

read_band_parameters() (*in module interface*), 117
readbandparam() (*in module inputoutput*), 112
readcellparam() (*in module inputoutput*), 112
readparam() (*in module inputoutput*), 113
real_to_rec() (*lattice.Lattice method*), 86
rec_to_real() (*lattice.Lattice method*), 86
regularcell() (*lattice.Lattice method*), 87

S

scattering (*module*), 48
scattering_dos() (*in module scattering*), 50
scattering_parabolic() (*in module scattering*), 52
scipy_e_integrals() (*in module lbteint*), 63
scipy_k_integrals() (*in module lbteint*), 63
scipy_k_integrals_discrete() (*in module lbteint*), 64
scipy_k_integrals_discrete2() (*in module lbteint*), 64
setup_scattering() (*transport.Transport method*), 57
skw_warning() (*in module inputoutput*), 113
spglib_error() (*in module inputoutput*), 113
start_message() (*in module inputoutput*), 113

T

t4me (*module*), 54
Transport (*class in transport*), 54
transport (*module*), 54

U

utils (*module*), 118

W

wildmagic_warning() (*in module inputoutput*), 113