ThermoElectric Documentation

ThermoElectric

Mar 28, 2022

CONTENTS:

1 Installation **3**

1.1 Getting Started 3

1.2 References 6

1.3 API Documentation 7

Index **19**

ThermoElectric is a computational framework that computes electron transport coefficients with unique features to design the nanoscale morphology of thermoelectric (TE) materials to obtain electron scattering that will enhance performance through electron energy filtering. The code uses the linear Boltzmann transport equation to compute the electrical properties of electrical conductivity, Seebeck coefficient, electron thermal conductivity, etc., under relaxation time approximation.

CHAPTER

ONE

INSTALLATION

```
gh repo clone ariahosseini/ThermoElectric
cd ThermoElectric
sudo python setup.py install
```

1.1 Getting Started

This page details how to get started with ThermoElectric.

1.1.1 Intrinsic Properties

Once installed, you can use the package. This example shows how to model the electron transport coefficients in nano-structured silicon.

The following block of code computes the energy range [eV], temperature range [K], and the electronic band gap [eV]:

```
import ThermoElectric as TE
import numpy as np
from numpy.linalg import norm
from scipy.interpolate import PchipInterpolator as interpolator

energy_min = 0.0  # Minimum energy level [eV]
energy_max = 1  # Maximum energy level [eV]
num_enrg_sample = num_enrg_sample = 4000  # Number of energy points

tmpr_min = 300  # Minimum temperature [K]
tmpr_max = 1300  # Maximum temperature [K]
tmpr_step = 50  # Number of temperature points

engr = TE.energy_range(energy_min = energy_min, energy_max = energy_max,
                       sample_size = num_enrg_sample)
tmpr = TE.temperature(temp_min = tmpr_min, temp_max = tmpr_max, del_temp = tmpr_step)
electronic_gap = TE.band_gap(Eg_o = 1.17, Ao = 4.73e-4, Bo = 636, temp = tmpr)
```

ThermoElectric uses the following form for the band gap:

$$\mathrm{E_g(T)} = \mathrm{E_g(0)} - \frac{\mathrm{A_o T^2}}{\mathrm{T + B_o}}$$

For the silicon, $\mathrm{E_g(T)} = 1.17$ [eV], $\mathrm{A_o} = 4.73 \times 10^{-4}$ [eV/K], $\mathrm{B_o} = 636$ [K], are used. For more details, see "Properties of Advanced Semiconductor Materials" by Michael E. Levinshtein.

Next step is to compute total carrier concentration.

```
carrier_con = TE.carrier_concentration(path_extrinsic_carrier =
                                       'Exp_Data/experimental-carrier-concentration-5pct-
↪direction-up.txt',
                                       band_gap = electronic_gap, Ao = 5.3e21, Bo = 3.
↪5e21, temp = tmpr)
```

The intrinsic carrier concentration is computed using $N_i = \sqrt{N_c N_v}\exp(\frac{E_g}{2k_B T})$, where $N_c = A_o T^{3/2}$ and $N_v = B_o T^{3/2}$ are the effective densities of states in the conduction and valence bands, respectively. For the silicon, $A_o = 5.3 \times 10^{21}$ $[m^{-3}K^{-3/2}]$, $B_o = 3.5 \times 10^{21}$ $[m^{-3}K^{-3/2}]$, are used from "Properties of Advanced Semiconductor Materials" by Michael E. Levinshtein.

We need to define the reciprocal space basis. For Si, the basis is defined as:

```
lattice_parameter = 5.40e-10  # Si lattice parameter in [m]
lattice_vec = np.array([[1,1,0],[0,1,1],[1,0,1]])*lattice_parameter/2 # lattice vector␣
↪in [1/m]
a_rp = np.cross(lattice_vec[1], lattice_vec[2])/ np.dot(lattice_vec[0], np.cross(lattice_
↪vec[1], lattice_vec[2]))
b_rp = np.cross(lattice_vec[2], lattice_vec[0])/ np.dot(lattice_vec[1], np.cross(lattice_
↪vec[2], lattice_vec[0]))
c_rp = np.cross(lattice_vec[0], lattice_vec[1])/ np.dot(lattice_vec[2], np.cross(lattice_
↪vec[0], lattice_vec[1]))
recip_lattice_vec = np.array([a_rp, b_rp, c_rp])  # Reciprocal lattice vectors
```

Next, we compute the band structure [eV], group velocity [m/s], and the density of states ($1/m^3$)

```
num_kpoints = 800  # Number of kpoints in EIGENVAL
num_dos = 2000 # Number of points in DOSCAR
num_bands = 8  # Number of bands in Si
num_qpoints = 200  # Number of q-points in desired band
valley_idx = 1118  # The index of valley in DOSCAR
unitcell_vol = 2*19.70272e-30 # Silicon unitcell volume

dispersion = TE.band_structure(path_eigen = 'DFT_Data/EIGENVAL', skip_lines = 6, num_
↪bands = num_bands,
                               num_kpoints = num_kpoints)
kp = dispersion['k_points']
band_struc = dispersion['electron_dispersion']

band_dir = band_str[400: 400 + num_qpoints, 4]  # The forth column is the conduction band
min_band = np.argmin(band_dir, axis=0)  # The index of the conduction band valley
max_band = np.argmax(band_dir, axis=0)  # The index of the maximum energy level in the␣
↪conduction band

kp_rl = 2 * np.pi * kp @ RLv  # Wave-vectors in the reciprocal space
kp_mag = norm(kp_rl, axis=1)  # The magnitude of the wave-vectors
kp_engr = kp_mag[400+max_band: 400+min_band]

energy_kp = band_struc[400+max_band: 400+min_band, 4] - band_str[400+min_band, 4]
sort_enrg = np.argsort(energy_kp, axis=0)
# The electron group velocity
grp_velocity = TE.group_velocity(kpoints = kp_engr[sort_enrg], energy_kp = energy_
↪kp[sort_enrg], energy = engr)
```

(continues on next page)

(continued from previous page)

```
# The electronic density of states
e_density = TE.electron_density(path_density = 'DFT_Data/DOSCAR', header_lines = 6,␣
↪unitcell_volume= unitcell_vol,
                                num_dos_points = num_dos, valley_point = valley_idx,␣
↪energy = engr)
```

1.1.2 Fermi Energy Level

We can estimate the Fermi energy level using Joyce Dixon approximation

```
joyce_dixon = TE.fermi_level(carrier = carrier_con, energy = engr, density = e_density,␣
↪Nc = None,
                             Ao = 5.3e21, temp = tmpr)
```

Joyce Dixon approximate the Fermi level using $\mathrm{E_f} = \ln\left(\frac{\mathrm{N_i}}{\mathrm{Nc}}\right) + \frac{1}{\sqrt{8}}\left(\frac{\mathrm{N_i}}{\mathrm{Nc}}\right) - \left(\frac{3}{16} - \frac{\sqrt{3}}{9}\right)\left(\frac{\mathrm{N_i}}{\mathrm{Nc}}\right)^2$.

Next, we are using a self-consistent algorithm to accurately compute the Fermi level.

```
fermi = TE.fermi_self_consistent(carrier = carrier_con, temp = tmpr, energy= engr,␣
↪density= e_density,
                                 fermi_levels = joyce_dixon)
```

Fermi distribution and its derivative (Fermi window) are computed as

```
k_bolt = 8.617330350e-5  # Boltzmann constant in [eV/K]
fermi_dist = TE.fermi_distribution(energy = engr, fermi_level = fermi[1][np.newaxis, :],␣
↪temp = tmpr)
np.savetxt("Matlab_Files/Ef.out", fermi[1] / tmpr / k_bolt)
```

We need Ef.out to compute the -0.5-order and 0.5-order Fermi-Dirac integral. The fermi.m is an script writen by Natarajan and Mohankumar that may be used to evaluate the half-order Fermi-Dirac integral integrals. An alternative python tool is dfint

```
pip install fdint
```

The generalized Debye length is computed as $L_D = \frac{e^2 N_c}{4\pi\epsilon\epsilon_o k_B T}\left[F_{-1/2}(\eta) + \frac{15\alpha k_B T}{4} F_{1/2}(\eta)\right]$

```
eps_o = 8.854187817e-12  # Permittivity in vacuum F/m
mass_e = 9.109e-31  # Electron rest mass in Kg
h_bar = 6.582119e-16  # Reduced Planck constant in eV.s
e2C = 1.6021765e-19  # e to Coulomb unit change
nonparabolic_term = 0.5  # Non-parabolic term
dielectric = 11.7  # Relative dielectricity

mass_cond = 0.23 * mass_e * (1 + 5 * nonparabolic_term * k_bolt * tmpr)  # Conduction␣
↪band effective mass
Nc = 2*(mass_cond * k_bolt * tmpr / h_bar**2 / 2/ np.pi/ e2C)**(3./2)
fermi_ints = np.loadtxt("Matlab_Files/fermi-5pct-dir-up.out", delimiter=',')
screen_len = np.sqrt(1 / (Nc / dielectric / eps_o / k_bolt / tmpr * e2C *
                    (fermi_ints[1] + 15 * nonparabolic_term * k_bolt * tmpr / 4 * fermi_
↪ints[0])))
```

1.1.3 Electron Lifetime

The electron-phonon and electron-impurity scattering rates are computed as

```
bulk_module = 98  # Bulk module (GPA)
rho = 2329  # Mass density (Kg/m3)
speed_sound = np.sqrt(bulk_module/rho)  # Speed of sound
num_vally = 6

tau_ph = TE.tau_p(energy = engr, alpha_term = nonparabolic_term , D_v = 2.94, D_a = 9.5,␣
↪temp = tmpr,
                  vel_sound = speed_sound, DoS = e_density, rho = rho)
tau_imp = TE.tau_strongly_screened_coulomb(DoS = e_density, screen_len = screen_len, n_
↪imp = carrier_con,
                                           dielectric = dielectric)
tau = TE.matthiessen(engr, num_vally * tau_ph['nonparabolic_ph_lifetime'], num_vally *␣
↪tau_imp)
```

The following equations are used

$$\tau_{\mathrm{p}}(\mathrm{E}) = \frac{\rho \nu_{\mathrm{s}}^2 \hbar}{\pi \Phi_{\mathrm{A}}^2 \mathrm{k_B T D(E)}} \left(\left[1 - \frac{\alpha \mathrm{E}}{1 + 2\alpha \mathrm{E}} \left(1 - \frac{\Phi_{\mathrm{v}}}{\Phi_{\mathrm{A}}} \right) \right]^2 - \frac{8}{3} \frac{\alpha \mathrm{E}(1 + \alpha \mathrm{E})}{(1 + 2\alpha \mathrm{E})^2} \frac{\mathrm{D_v}}{\mathrm{D_A}} \right)^{-1}$$

$$\tau_{\mathrm{ion}}(\mathrm{E}) = \frac{\hbar}{\pi \mathrm{N_i} \left(\frac{\mathrm{e}^2 \mathrm{L_D^2}}{4\pi\epsilon\epsilon_{\mathrm{o}}} \right)^2 \mathrm{D(E)}}$$

The Matthiessen's rule is used to add the scattering rates.

1.1.4 Electrical Properties

Finally the electrical properties are computed using

```
elec_propt = TE.electrical_properties(E = engr, DoS = e_density, vg = grp_velocity, Ef =␣
↪fermi[1][np.newaxis, :],
                                      dfdE = fermi_dist[1], temp = tmpr, tau = tau)
```

1.2 References

If you use ThermoElectric, please consider citing:

```
@article{doi:10.1021/acsaem.0c02640,
    author = {Hosseini, S. Aria and Romano, Giuseppe and Greaney, P. Alex},
    title = {Mitigating the Effect of Nanoscale Porosity on Thermoelectric Power Factor␣
↪of Si},
    journal = {ACS Applied Energy Materials},
    volume = {4},
    number = {2},
    pages = {1915-1923},
    year = {2021},
    doi = {10.1021/acsaem.0c02640},
```

(continues on next page)

(continued from previous page)

```
    url = {https://doi.org/10.1021/acsaem.0c02640},
    eprint = {https://doi.org/10.1021/acsaem.0c02640}
    }
```

```
@Article{nano11102591,
    AUTHOR = {Hosseini, S. Aria and Romano, Giuseppe and Greaney, P. Alex},
    TITLE = {Enhanced Thermoelectric Performance of Polycrystalline Si0.8Ge0.2 Alloys␣
↪through the Addition of Nanoscale Porosity},
    JOURNAL = {Nanomaterials},
    VOLUME = {11},
    YEAR = {2021},
    NUMBER = {10},
    ARTICLE-NUMBER = {2591},
    URL = {https://www.mdpi.com/2079-4991/11/10/2591},
    PubMedID = {34685032},
    ISSN = {2079-4991},
    DOI = {10.3390/nano11102591}
    }
```

```
@article{hosseini2021enhanced,
    title={Enhanced Thermoelectric ZT in the Tails of the Fermi Distribution via␣
↪Electron Filtering by Nanoinclusions - Model Electron Transport in Nanocomposites},
    author={Hosseini, S Aria and Coleman, Devin and Bux, Sabah and Greaney, P Alex and␣
↪Mangolini, Lorenzo},
    journal={arXiv preprint arXiv:2110.13375},
    year={2021}
    }
```

1.3 API Documentation

ThermoElectric.energy_range(energy_min, ...)	Create a 2D array of energy space sampling
ThermoElectric.temperature([temp_min, ...])	Create a 2D array of temperature sampling
ThermoElectric.fermi_distribution(energy, ...)	A function to compute the Fermi distribution and the Fermi window — the first energy derivative of the Fermi level
ThermoElectric.matthiessen(energy, *args)	A function to compute the total lifetime using Matthiessen's rule
ThermoElectric.kpoints(path2kpoints[, ...])	Create a 2D array of temperature sampling
ThermoElectric.carrier_concentration(...[, ...])	This function computes the carrier concentration.
ThermoElectric.band_structure(path_eigen, ...)	A function to read "EIGENVAL" file
ThermoElectric.electron_density(...)	A function to read "DOSCAR" file
ThermoElectric.band_gap(Eg_o, Ao, Bo[, temp])	This method uses Eg(T)=Eg(T=0)-Ao*T**2/(T+Bo) to approximate the temperature dependency of the dielectrics band gap.
ThermoElectric.analytical_dos(range_energy, ...)	This function approximate the electron density of state for parabolic and non-parabolic bands

continues on next page

Table 1 – continued from previous page

ThermoElectric.fermi_level(carrier, energy, ...)	This function uses Joice Dixon approximation to predict Ef and thereby the carrier concentration at each temperature A good reference book is "Principles of Semiconductor Devices" by Sima Dimitrijev.
ThermoElectric.fermi_self_consistent(...)	Self-consistent calculation of the Fermi level from a given carrier concentration using Joyce Dixon approximation as the initial guess for degenerate semiconductors.
ThermoElectric.group_velocity(kpoints, ...)	Group velocity is the derivation of band structure from DFT.
ThermoElectric.analytical_group_velocity(...)	This method approximate the group velocity near the conduction band edge in case no DFT calculation is available.
ThermoElectric.tau_p(energy, alpha_term, ...)	Electron-phonon scattering rate using Ravich model
ThermoElectric.tau_strongly_screened_coulomb(...)	Electron-impurity scattering model in highly doped dielectrics
ThermoElectric.tau_screened_coulomb(energy, ...)	Electron-ion scattering rate — Brook-Herring model
ThermoElectric.tau_unscreened_coulomb(...)	Electron-ion scattering rate for shallow dopants ~10^18 1/cm^3 (no screening effect is considered)
ThermoElectric.tau_2d_cylinder(energy, ...)	A fast algorithm that uses Fermi's golden rule to compute the energy dependent electron scattering rate from cylindrical nano-particles or nano-scale pores infinitely extended perpendicular to the current.
ThermoElectric.tau3D_spherical(num_kpoints, ...)	A fast algorithm that uses Fermi's golden rule to compute the energy dependent electron scattering rate from spherical nano-particles or nano-scale pores.
ThermoElectric.electrical_properties(E, DoS, ...)	This function returns electronic properties.
ThermoElectric.filtering_effect(Uo, E, DoS, ...)	This function returns electric properties for the ideal filtering — where all the electrons up to a cutoff energy level of Uo are completely hindered.
ThermoElectric.phenomenological(Uo, tau_o, ...)	This function returns electric properties for the phenomenological filtering — where a frequency independent lifetime of tau_o is imposed to all the electrons up to a cutoff energy level of Uo

1.3.1 ThermoElectric.energy_range

ThermoElectric.**energy_range**(*energy_min: float*, *energy_max: float*, *sample_size: int*) → numpy.ndarray

Create a 2D array of energy space sampling

Parameters

- **energy_min** (*float*) – minimum energy level of electron in conduction band
- **energy_max** (*float*) – maximum energy level of electron in conduction band
- **sample_size** (*int*) – Sampling size of the energy space

Returns **energy_sample** – Energy sampling with the size of [1, sample_size]

Return type np.ndarray

1.3.2 ThermoElectric.temperature

ThermoElectric.**temperature**(*temp_min: float = 300*, *temp_max: float = 1301*, *del_temp: float = 100*) → numpy.ndarray

Create a 2D array of temperature sampling

Parameters

- **temp_min** (*float*) – minimum temperature
- **temp_max** (*float*) – maximum temperature
- **del_temp** (*float*) – Sampling size of the temperature range

Returns **temp_range** – Temperature sampling with the size of [1, del_temp]

Return type np.ndarray

1.3.3 ThermoElectric.fermi_distribution

ThermoElectric.**fermi_distribution**(*energy: numpy.ndarray*, *fermi_level: numpy.ndarray*, *temp: Optional[numpy.ndarray] = None*) → numpy.ndarray

A function to compute the Fermi distribution and the Fermi window — the first energy derivative of the Fermi level

Parameters

- **energy** (*np.ndarray*) – Energy range in conduction band
- **fermi_level** (*np.ndarray*) – Fermi level
- **temp** (*np.ndarray*) – Temperature range

Returns **fermi** – The first row is the Fermi distribution and the second row is the derivative (Fermi window)

Return type np.ndarray

1.3.4 ThermoElectric.matthiessen

ThermoElectric.**matthiessen**(*energy: numpy.ndarray*, **args*) → numpy.ndarray

A function to compute the total lifetime using Matthiessen's rule

Parameters

- **energy** (*np.ndarray*) – Energy levels
- ***args** (*np.ndarray*) – Electron lifetime from different scattering sources

Returns **tau** – The total electron lifetime

Return type np.ndarray

1.3.5 ThermoElectric.kpoints

ThermoElectric.**kpoints**(*path2kpoints: str*, *delimiter: Optional[str] = None*, *skip_rows: int = 0*) → numpy.ndarray

Create a 2D array of temperature sampling

Parameters

- **path2kpoints** (*str*) – Path to kpoints file
- **delimiter** (*str*) – Default it None for ,
- **skip_rows** (*int*) – Number of lines to skip, default is 0

Returns **wave_points** – Wave vectors

Return type np.ndarray

1.3.6 ThermoElectric.carrier_concentration

ThermoElectric.**carrier_concentration**(*path_extrinsic_carrier: str*, *band_gap: numpy.ndarray*, *Ao: Optional[float] = None*, *Bo: Optional[float] = None*, *Nc: Optional[float] = None*, *Nv: Optional[float] = None*, *temp: Optional[numpy.ndarray] = None*) → numpy.ndarray

This function computes the carrier concentration. The extrinsic carrier concentration is from experiments. The following formula is used to compute intrinsic carrier concentration: n = sqrt(Nc*Nv)*exp(-Eg/kB/T/2) A good reference book is "Principles of Semiconductor Devices" by Sima Dimitrijev

Parameters

- **path_extrinsic_carrier** (*str*) – Path to kpoints file
- **band_gap** (*np.ndarray*) – The electronic band gap
- **Ao** (*float*) – Experimentally fitted parameter (Nc ~ Ao*T^(3/2))
- **Bo** (*float*) – Experimentally fitted parameter (Nv ~ Ao*T^(3/2))
- **Nc** (*float*) – The effective densities of states in the conduction band
- **Nv** (*float*) – The effective densities of states in the conduction band
- **temp** (*np.ndarray*) – Temperature range

Returns **carrier** – The total carrier concentration

Return type np.ndarray

1.3.7 ThermoElectric.band_structure

ThermoElectric.**band_structure**(*path_eigen: str*, *skip_lines: int*, *num_bands: int*, *num_kpoints: int*) → dict

A function to read "EIGENVAL" file

Parameters

- **path_eigen** (*str*) – Path to EIGENVAL file
- **skip_lines** (*int*) – Number of lines to skip
- **num_bands** (*int*) – Number of bands
- **num_kpoints** (*int*) – number of wave vectors

Returns **dispersion** – Band structure

Return type dict

1.3.8 ThermoElectric.electron_density

ThermoElectric.**electron_density**(*path_density: str*, *header_lines: int*, *num_dos_points: int*, *unitcell_volume: float*, *valley_point: int*, *energy: numpy.ndarray*) → numpy.ndarray

A function to read "DOSCAR" file

Parameters

- **path_density** (*str*) – Path to DOSCAR file
- **header_lines** (*int*) – Number of lines to skip
- **num_dos_points** (*int*) – Number of points in DOSCAR
- **unitcell_volume** (*float*) – The unit cell volume is in [m]
- **valley_point** (*int*) – Where valley is located in DOSCAR
- **energy** (*np.ndarray*) – The energy range

Returns **density** – Electron density of states

Return type np.ndarray

1.3.9 ThermoElectric.band_gap

ThermoElectric.**band_gap**(*Eg_o: float*, *Ao: float*, *Bo: float*, *temp: Optional[numpy.ndarray] = None*) → numpy.ndarray

This method uses Eg(T)=Eg(T=0)-Ao*T**2/(T+Bo) to approximate the temperature dependency of the dielectrics band gap. A good reference is "Properties of Advanced Semiconductor Materials" by Michael E. Levinshtein.

Parameters

- **Eg_o** (*float*) – The band gap at zero Kelvin
- **Ao** (*float*) – Experimentally fitted parameter
- **Bo** (*float*) – Experimentally fitted parameter
- **temp** (*np.ndarray*) – Temperature range

Returns **Eg** – Temperature-dependent band gap

Return type np.ndarray

1.3.10 ThermoElectric.analytical_dos

ThermoElectric.**analytical_dos**(*range_energy: numpy.ndarray*, *electron_eff_mass: float*, *nonparabolic_term: numpy.ndarray*) → dict

This function approximate the electron density of state for parabolic and non-parabolic bands in case DFT calculation is not available.

Parameters

- **range_energy** (*np.ndarray*) – Electron energy range
- **electron_eff_mass** (*float*) – Electron effective mass
- **nonparabolic_term** (*np.ndarray*) – Non-parabolic term (shows the mixture of S and P orbitals)

Returns **DoS** – The DoS['DoS_nonparabolic'] is phonon density of states for non-parabolic band and the DoS['DoS_parabolic'] is the phonon density of states for non-parabolic.

Return type dict

1.3.11 ThermoElectric.fermi_level

ThermoElectric.**fermi_level**(*carrier: numpy.ndarray*, *energy: numpy.ndarray*, *density: numpy.ndarray*, *Nc: Optional[float] = None*, *Ao: Optional[float] = None*, *temp: Optional[numpy.ndarray] = None*) → numpy.ndarray

This function uses Joice Dixon approximation to predict Ef and thereby the carrier concentration at each temperature A good reference book is "Principles of Semiconductor Devices" by Sima Dimitrijev.

Parameters

- **carrier** (*np.ndarray*) – Total carrier concentration
- **energy** (*np.ndarray*) – The electron energy level
- **density** (*np.ndarray*) – The electron density of states
- **Nc** (*float*) – The effective densities of states in the conduction band
- **Ao** (*float*) – Experimentally fitted parameter (Nc ~ Ao*T^(3/2))
- **temp** (*np.ndarray*) – Temperature range

Returns **output** – The first row is the carrier concentration and the second one is the Fermi level

Return type np.ndarray

1.3.12 ThermoElectric.fermi_self_consistent

ThermoElectric.**fermi_self_consistent**(*carrier: numpy.ndarray*, *temp: numpy.ndarray*, *energy: numpy.ndarray*, *density: numpy.ndarray*, *fermi_levels: numpy.ndarray*) → numpy.ndarray

Self-consistent calculation of the Fermi level from a given carrier concentration using Joyce Dixon approximation as the initial guess for degenerate semiconductors. As a default value of 4000 sampling points in energy range from Ef(JD)-0.4 eV up to Ef(JD)+0.2 is considered.This looks reasonable in most cases. The index is printed out if it reaches the extreme index of (0) or (4000), increase energy range.

Parameters

- **carrier** (*np.ndarray*) – Total carrier concentration
- **energy** (*np.ndarray*) – The electron energy level
- **density** (*np.ndarray*) – The electron density of states
- **fermi_levels** (*np.ndarray*) – Joyce Dixon femi level approximation as the initial guess
- **temp** (*np.ndarray*) – Temperature range

Returns **output** – The first row is the carrier concentration and the second one is the Fermi level

Return type np.ndarray

1.3.13 ThermoElectric.group_velocity

ThermoElectric.**group_velocity**(*kpoints: numpy.ndarray*, *energy_kp: numpy.ndarray*, *energy: numpy.ndarray*) → numpy.ndarray

Group velocity is the derivation of band structure from DFT. Linear BTE needs single band data. Reciprocal lattice vector is required

Parameters

- **kpoints** (*np.ndarray*) – Wave vectors
- **energy_kp** (*np.ndarray*) – Energy for each wave vector
- **energy** (*np.ndarray*) – Energy range

Returns **velocity** – Group velocity

Return type np.ndarray

1.3.14 ThermoElectric.analytical_group_velocity

ThermoElectric.**analytical_group_velocity**(*energy: numpy.ndarray*, *lattice_parameter: numpy.ndarray*, *num_kpoints: numpy.ndarray*, *m_eff: numpy.ndarray*, *valley: numpy.ndarray*, *dk_len: numpy.ndarray*, *alpha_term: numpy.ndarray*) → numpy.ndarray

This method approximate the group velocity near the conduction band edge in case no DFT calculation is available.

Parameters

- **energy** (*np.ndarray*) – Energy range
- **lattice_parameter** (*np.ndarray*) – Lattice parameter
- **num_kpoints** (*np.ndarray*) – Number of wave vectors
- **m_eff** (*np.ndarray*) – Effective mass along different axis
- **valley** (*np.ndarray*) – Conduction valley
- **dk_len** (*np.ndarray*) – magnitude of wave vectors
- **alpha_term** (*np.ndarray*) – Non-parabolic term

Returns **velocity** – Group velocity

Return type np.ndarray

1.3.15 ThermoElectric.tau_p

ThermoElectric.**tau_p**(*energy: numpy.ndarray*, *alpha_term: numpy.ndarray*, *D_v: float*, *D_a: float*, *temp: numpy.ndarray*, *vel_sound: float*, *DoS: numpy.ndarray*, *rho: float*) → dict

Electron-phonon scattering rate using Ravich model

Parameters

- **energy** (*np.ndarray*) – Energy range
- **alpha_term** (*np.ndarray*) – Non-parabolic term
- **D_v** (*float*) – Hole deformation potential
- **D_a** (*float*) – Electron deformation potential
- **temp** (*np.ndarray*) – Temperature
- **vel_sound** (*float*) – Sound velocity
- **DoS** (*np.ndarray*) – Density of state
- **rho** (*float*) – Mass density

Returns **output** – parabolic and non-parabolic electron-phonon lifetime

Return type dict

1.3.16 ThermoElectric.tau_strongly_screened_coulomb

ThermoElectric.**tau_strongly_screened_coulomb**(*DoS: numpy.ndarray*, *screen_len: numpy.ndarray*, *n_imp: numpy.ndarray*, *dielectric: float*) → numpy.ndarray

Electron-impurity scattering model in highly doped dielectrics

Note that for highly doped semiconductors, screen length plays a significant role, therefor should be computed carefully. Highly suggest to use following matlab file "Fermi.m" from: https://www.mathworks.com/matlabcentral/fileexchange/13616-fermi

If committed to use python, the package "dfint" works with python2 pip install fdint

Parameters

- **DoS** (*np.ndarray*) – Density of states
- **screen_len** (*np.ndarray*) – Screening length
- **n_imp** (*np.ndarray*) – impurity scattering
- **dielectric** (*float*) – Dielectric constant

Returns **tau** – Electron-impurity lifetime

Return type np.ndarray

1.3.17 ThermoElectric.tau_screened_coulomb

ThermoElectric.**tau_screened_coulomb**(*energy: numpy.ndarray*, *mass_c: numpy.ndarray*, *screen_len: numpy.ndarray*, *n_imp: numpy.ndarray*, *dielectric: float*) → numpy.ndarray

Electron-ion scattering rate — Brook-Herring model

Note that for highly doped semiconductors, screen length plays a significant role, therefor should be computed carefully. Highly suggest to use following matlab file "Fermi.m" from: https://www.mathworks.com/matlabcentral/fileexchange/13616-fermi

If committed to use python, the package "dfint" works with python2 pip install fdint

Parameters

- **energy** (*np.ndarray*) – Energy range
- **mass_c** (*np.ndarray*) – Conduction band effective mass
- **screen_len** (*np.ndarray*) – Screening length
- **n_imp** (*np.ndarray*) – impurity scattering
- **dielectric** (*float*) – Dielectric constant

Returns **tau** – Electron-impurity lifetime

Return type np.ndarray

1.3.18 ThermoElectric.tau_unscreened_coulomb

ThermoElectric.**tau_unscreened_coulomb**(*energy: numpy.ndarray*, *mass_c: numpy.ndarray*, *n_imp: numpy.ndarray*, *dielectric: float*) → numpy.ndarray

Electron-ion scattering rate for shallow dopants ~10^18 1/cm^3 (no screening effect is considered)

Parameters

- **energy** (*np.ndarray*) – Energy range
- **mass_c** (*np.ndarray*) – Conduction band effective mass
- **n_imp** (*np.ndarray*) – impurity scattering
- **dielectric** (*float*) – Dielectric constant

Returns **tau** – Electron-impurity lifetime

Return type np.ndarray

1.3.19 ThermoElectric.tau_2d_cylinder

ThermoElectric.**tau_2d_cylinder**(*energy: numpy.ndarray*, *num_kpoints: numpy.ndarray*, *Uo: float*, *relative_mass: numpy.ndarray*, *volume_frac: float*, *valley: numpy.ndarray*, *dk_len: float*, *ro: numpy.ndarray*, *lattice_parameter: float*, *n_sample=2000*) → numpy.ndarray

A fast algorithm that uses Fermi's golden rule to compute the energy dependent electron scattering rate from cylindrical nano-particles or nano-scale pores infinitely extended perpendicular to the current.

Parameters

- **energy** (*np.ndarray*) – Energy range

- **num_kpoints** (*np.ndarray*) – Number of kpoints in each direction
- **Uo** (*float*) – Barrier height
- **relative_mass** (*np.ndarray*) – Relative mass of electron
- **volume_frac** (*float*) – Defects volume fraction
- **valley** (*np.ndarray*) – Conduction band valley indices
- **dk_len** (*float*) – Sample size
- **ro** (*np.ndarray*) – Cylinder radius
- **lattice_parameter** (*float*) – lattice parameter
- **n_sample** (*int*) – Mesh sample size

Returns **tau_cylinder** – Electron-defect lifetime

Return type np.ndarray

1.3.20 ThermoElectric.tau3D_spherical

ThermoElectric.**tau3D_spherical**(*num_kpoints: numpy.ndarray*, *Uo: float*, *relative_mass: numpy.ndarray*, *volume_frac: float*, *valley: numpy.ndarray*, *dk_len: float*, *ro: numpy.ndarray*, *lattice_parameter: float*, *n_sample=32*) → numpy.ndarray

A fast algorithm that uses Fermi's golden rule to compute the energy dependent electron scattering rate from spherical nano-particles or nano-scale pores.

Parameters

- **num_kpoints** (*np.ndarray*) – Number of kpoints in each direction
- **Uo** (*float*) – Barrier height
- **relative_mass** (*np.ndarray*) – Relative mass of electron
- **volume_frac** (*float*) – Defects volume fraction
- **valley** (*np.ndarray*) – Conduction band valley indices
- **dk_len** (*float*) – Sample size
- **ro** (*np.ndarray*) – Cylinder radius
- **lattice_parameter** (*float*) – lattice parameter
- **n_sample** (*int*) – Mesh sample size

Returns **tau** – Electron-defect lifetime

Return type np.ndarray

1.3.21 ThermoElectric.electrical_properties

ThermoElectric.**electrical_properties**(*E: numpy.ndarray*, *DoS: numpy.ndarray*, *vg: numpy.ndarray*, *Ef: numpy.ndarray*, *dfdE: numpy.ndarray*, *temp: numpy.ndarray*, *tau: numpy.ndarray*) → dict

This function returns electronic properties. Good references on this topic are: "Near-equilibrium Transport: Fundamentals And Applications" by Changwook Jeong and Mark S. Lundstrom "Nanoscale Energy Transport and Conversion: A Parallel Treatment of Electrons, Molecules, Phonons, and Photons" by Gang Chen.

Parameters

- **E** (*np.ndarray*) – Energy range
- **DoS** (*np.ndarray*) – Electron density of state
- **vg** (*np.ndarray*) – Group velocity
- **Ef** (*np.ndarray*) – Fermi level
- **dfdE** (*np.ndarray*) – Fermi window
- **temp** (*np.ndarray*) – Temperature range
- **tau** (*np.ndarray*) – Lifetime

Returns **coefficients** – Linear BTE prediction of electrical properties

Return type dict

1.3.22 ThermoElectric.filtering_effect

ThermoElectric.**filtering_effect**(*Uo*, *E*, *DoS*, *vg*, *Ef*, *dfdE*, *temp*, *tau_bulk*)

This function returns electric properties for the ideal filtering — where all the electrons up to a cutoff energy level of Uo are completely hindered.

Parameters

- **Uo** (*np.ndarray*) – Barrier height
- **E** (*np.ndarray*) – Energy range
- **DoS** (*np.ndarray*) – Electron density of state
- **vg** (*np.ndarray*) – Group velocity
- **Ef** (*np.ndarray*) – Fermi level
- **dfdE** (*np.ndarray*) – Fermi window
- **temp** (*np.ndarray*) – Temperature range
- **tau_bulk** (*np.ndarray*) – Lifetime in bulk material

Returns **output** – Linear BTE prediction of electrical properties

Return type dict

1.3.23 ThermoElectric.phenomenological

ThermoElectric.**phenomenological**(*Uo*, *tau_o*, *E*, *DoS*, *vg*, *Ef*, *dfdE*, *temp*, *tau_bulk*)

This function returns electric properties for the phenomenological filtering — where a frequency independent lifetime of tau_o is imposed to all the electrons up to a cutoff energy level of Uo

Parameters

- **Uo** (*np.ndarray*) – Barrier height
- **tau_o** (*np.ndarray*) – Phenomenological lifetime
- **E** (*np.ndarray*) – Energy range
- **DoS** (*np.ndarray*) – Electron density of state
- **vg** (*np.ndarray*) – Group velocity
- **Ef** (*np.ndarray*) – Fermi level
- **dfdE** (*np.ndarray*) – Fermi window
- **temp** (*np.ndarray*) – Temperature range
- **tau_bulk** (*np.ndarray*) – Lifetime in bulk material

Returns **output** – Linear BTE prediction of electrical properties

Return type dict

INDEX

A

analytical_dos() (*in module ThermoElectric*), 12
analytical_group_velocity() (*in module ThermoElectric*), 13

B

band_gap() (*in module ThermoElectric*), 11
band_structure() (*in module ThermoElectric*), 10

C

carrier_concentration() (*in module ThermoElectric*), 10

E

electrical_properties() (*in module ThermoElectric*), 17
electron_density() (*in module ThermoElectric*), 11
energy_range() (*in module ThermoElectric*), 8

F

fermi_distribution() (*in module ThermoElectric*), 9
fermi_level() (*in module ThermoElectric*), 12
fermi_self_consistent() (*in module ThermoElectric*), 12
filtering_effect() (*in module ThermoElectric*), 17

G

group_velocity() (*in module ThermoElectric*), 13

K

kpoints() (*in module ThermoElectric*), 10

M

matthiessen() (*in module ThermoElectric*), 9

P

phenomenological() (*in module ThermoElectric*), 18

T

tau3D_spherical() (*in module ThermoElectric*), 16
tau_2d_cylinder() (*in module ThermoElectric*), 15
tau_p() (*in module ThermoElectric*), 14
tau_screened_coulomb() (*in module ThermoElectric*), 15
tau_strongly_screened_coulomb() (*in module ThermoElectric*), 14
tau_unscreened_coulomb() (*in module ThermoElectric*), 15
temperature() (*in module ThermoElectric*), 9