Octopost

Christian Kern

Oct 19, 2021

CONTENTS

Octopost contains a pool of python3 tools for post-processing data obtained with the TDDFT code OCTOPUS[1].

CHAPTER

ONE

CONTENTS

1.1 Octopost

Octopost contains a pool of python3 tools for post-processing data obtained with the TDDFT code OCTOPUS[1].

Octopost can be imported like any Python package into your analysis scripts and provides a set of easy-to-use methods for extracting, preparing and even plotting standard information like bandstructure, various (projected) density of states, reciprocal cell for periodic systems and so on. The main focus, however, lies in the ARPES (Angle-Resolved PhotoEmission Spectroscopy) methods which make calculating ARPES maps (also called k-maps) easy and quick. The calculated maps can be plotted and exported in a .hdf5 file compatible with the POT (Photoemssion Orbital Tomography) simulation and analysis software kMap.py[2].

GitLab Page: https://gitlab.com/ckern/octopost
Documentation: https://octopost.readthedocs.io/en/latest/

1.1.1 Authors

- Christian Kern, MSc. (christian.kern@uni-graz.at)
- Peter Puschnig, Assoz. Prof. Dipl.-Ing. Dr. (peter.puschnig@uni-graz.at)
- Dominik Brandstetter, BSc. (dominik.brandstetter@edu.uni-graz.at)

1.1.2 Quick-start

```
pip install octopost
```

1.1.3 Installation

It is generally recommended but not required to use a virtual enviroment like venv or conda for this package. For a tutorial on how to use them please see here.

User

As a user you only need the subset of packages nessecary to run the package. The package will also not be installed in an editable form. The package will be added to your current enviroment and will be available throughout it.:

```
make install
```

on Linux, or:

```
pip install .
```

on Windows.

Developer & Maintainer

In order to develop for Octopost it is required to use the '-e' flag for pip to install in editable mode. Furthermore, it is recommended to install the 'dev' and 'test' packages as well. Simply use:

```
make install-dev
```

on Linux or:

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

on Windows.

1.1.4 Test

If you wish to run the test suite please make sure you have Octopost and its 'test' dependencies installed. The latter is automatically the case if you followed the 'Developer & Maintainer' section of the installation guide. If you installed the 'User' version please additionally run:

```
pip install .[test]
```

In order to run the test use:

```
pytest
```

On Linux you can also use:

```
make test
```

1.1.5 Demo

Inside the './demo' folder you can find a number of small scripts showcasing different Octopost functionalities and how to use them. Due to storage constraints the actual Octopus output files required for the scripts will not be available on GitLab, however, for most scripts the output plots can be found in the './demo/output' folder.

1.1.6 References

[1] Nicolas Tancogne-Dejean, Micael J. T. Oliveira, Xavier Andrade, Heiko Appel, Carlos H. Borca, Guillaume Le Breton, Florian Buchholz, Alberto Castro, Stefano Corni, Alfredo A. Correa, Umberto De Giovannini, Alain Delgado, Florian G. Eich, Johannes Flick, Gabriel Gil, Adrián Gomez, Nicole Helbig, Hannes Hübener, René Jestädt, Joaquim Jornet-Somoza, Ask H. Larsen, Irina V. Lebedeva, Martin Lüders, Miguel A. L. Marques, Sebastian T. Ohlmann, Silvio Pipolo, Markus Rampp, Carlo A. Rozzi, David A. Strubbe, Shunsuke A. Sato, Christian Schäfer, Iris Theophilou, Alicia Welden, and Angel Rubio , "Octopus, a computational framework for exploring light-driven phenomena and quantum dynamics in extended and finite systems", J. Chem. Phys. 152, 124119 (2020) https://doi.org/10.1063/1.5142502

[2] Dominik Brandstetter, Xiaosheng Yang, Daniel Lüftner, F. Stefan Tautz, and Peter Puschnig , "kMap.py: A Python program for simulation and data analysis in photoemission tomography", Computer Physics Communications 263, 107905 (2021) https://doi.org/10.1016/j.cpc.2021.107905

1.2 Code Documenation

1.2.1 ARPES

Calculations related to the ARPES maps.

This module contains the 'ARPESModel' class, which handles all things related to ARPES.

class arpes_model.**ARPESModel**(*octopost*)

ARPES model handles all things related to ARPES.

Should not be used directly by the user but called by the abstraction layer octopost class.

Parameters **octopost** (*object*) – An instance of the octopost class.

get_3D_arpes(*file='PES_velocity_map.vtk'*, *E_num=10*, *E_range=[- 10, 0]*, *dE_b=0*, *field='probe'*, *omega=None*, *out=False*, *out_file='output.hdf5'*)

Calculates (kx, ky) ARPES maps at certain binding energies from a PES_ARPES file.

Reads the PES ARPES file specified with 'file' to calculate 'E_num' ARPES maps. The maps are calculated at binding energies specified in the semi open interval [E_range[0], E_range[1]). The energy of the incident light field can be specified via omega or read from the 'parser.log' file by using the name 'field' the field was given. The data can be exported in a kMap.py compatible file 'out_file' using 'out'.

Parameters

- **file** (*str*) – Relative path from the root directory of the calculation to the PES velocity map file (.vtk file).
- **E_num** (*int*) – Number of energy cuts in the E_range interval.
- **E_range** (*list*) – A list of length 2 specifying the semi-open interval [E_range[0], E_range[1]) of binding energies used for the energy cuts. All binding energies have to be E <= 0 and the units expected are 'eV'.
- **dE_b** (*float*) – Integration over the open interval E_kin +- dE_b. In 'eV'.
- **field** (*str*) – Name the incident light field was given in the Octopus calculation. Is used to automatically find the photon energy of the incident light field. Can be overridden using 'omega'.
- **omega** (*float*) – The energy of the incoming photon in 'eV'. Default is 'None', where Octopost tries to find the omega itself by reading the 'parser.log' file and looking up the field named 'field'.

- **out** (*bool*) – Whether or not to export the maps to a .hdf5 file. The file will contain the axis information and is compatible with kMap.
- **out_file** (*str*) – If out is True the path to the output file relative to the root directory of the calculation.

Returns

- **nd.array** – A (E_num, resolution, resolution) shaped array. First index are the individual slices (even if E_num is 1). Second and third index are kx and ky respectively.
- **nd.array** – 1D array containing the individual binding energies (not kinetic energy of the e) used for the slices in 'eV'.
- **nd.array** – 1D array containing the kx (and therefore the identical ky) axis values in 1/Angstrom.

get_3D_pes(*file*, *plot=False*)

Returns photoemission intensity over the energy.

Parameters

- **file** (*str*) – Relative path from the root directory of the calculation to the PES velocity map file (.vtk file).
- **plot** (*bool*) – If len(integrate) != 2 this option will be ignored. Otherwise a basic line plot over the remaining dimension will be returned.

Returns

- **nd.array** – The intensity integrated over the kx and ky axes.
- **nd.array** – The energy axis in eV. fig, ax: If plot == True and integration over two dimensions, the handles for the matplotlib plot.

get_arpes_bandstructure(*file='PES_ARPES.path'*, *k_points=[]*, *labels=[]*, *omega=None*, *field='probe'*, *plot=True*, *plot_fermi=True*, *plot_points=True*, *axes=None*)

Reads the ARPES intensity for a path through the first Brillouin zone, i.e. the bandstructure.

Reads the PES_ARPES.path file which contains the intensity of the photoemission process for a path through the first Brillouin zone. In theory this should resemble the bandstructure.

Parameters

- **file** (*str*) – Relative path from the root directory of the calculation to the PES velocity map file (.vtk file).
- **k_points** (*list*) – A list of special points (e.g. '', 'K' or other) in terms of steps from the last point (same as in 'inp' file). Sum of all steps needs to be total number of steps, i.e. one less than total number of points. If list is empty octopost tries to find the path information from the parser.log.
- **labels** (*list*) – A list of strings with the same length as k_points. Specifies a label in the plot for each point in k_points. Use empty string if no label (and line) is required.
- **omega** (*float*) – The photon energy in eV. If None is passed octopost tries to find the energy from parser.log.
- **field** (*str*) – Name of the external field octopost tries to find the photon energy by if not explicitly passed.
- **plot** (*bool*) – Whether to return a basic plot.
- **plot_fermi** (*bool*) – Plot a dashed line indicating the fermi energy.

- **plot_points** (*bool*) – Plot dashed line at each special point along the path that has a non empty label.
- **axes** (*fig, ax*) – Matplotlib handles to lay the plot on.

Returns

- **nd.array** – A nd.array with the k path through from the starting point in one over Angstrom.
- **nd.array** – 2D array with the intensity for each energy and k point.
- **fig, ax** – Matplotlib handles for the plot.

get_arpes_plot(*map_*, *k_axis*)

Convenience method to plot a ARPES map.

Uses the output of 'get_gasphase_arpes' to quickly plot an ARPES map.

Parameters

- **map** (*nd.array*) – 2D array containing the ARPES map.
- **k_axis** (*nd.array*) – 1D array containing the kx and ky axis in 1/A.

Returns **fig, ax** – Matplotlib handles for the plot.

get_gasphase_arpes(*file='PES_velocity_map.vtk'*, *E_num=10*, *E_range=[- 10, 0]*, *dE_b=0*, *field='probe'*, *omega=None*, *resolution=501*, *out=False*, *out_file='output.hdf5'*)

Calculates (kx, ky) ARPES maps at certain binding energies from a PES velocity file.

Reads the PES velocity file specified with 'file' to calculate 'E_num' ARPES maps. The maps are calculated at binding energies specified in the semi open interval [E_range[0], E_range[1]). The energy of the incident light field can be specified via omega or read from the 'parser.log' file by using the name 'field' the field was given. The resolution of each map is (resolution, resolution) and the data can be exported in a kMap.py compatible file 'out_file' using 'out'.

Parameters

- **file** (*str*) – Relative path from the root directory of the calculation to the PES velocity map file (.vtk file).
- **E_num** (*int*) – Number of energy cuts in the E_range interval.
- **E_range** (*list*) – A list of length 2 specifying the semi-open interval [E_range[0], E_range[1]) of binding energies used for the energy cuts. All binding energies have to be E <= 0 and the units expected are 'eV'.
- **dE_b** (*float*) – Integration over the open interval E_kin +- dE_b. In 'eV'.
- **field** (*str*) – Name the incident light field was given in the Octopus calculation. Is used to automatically find the photon energy of the incident light field. Can be overridden using 'omega'.
- **omega** (*float*) – The energy of the incoming photon in 'eV'. Default is 'None', where Octopost tries to find the omega itself by reading the 'parser.log' file and looking up the field named 'field'.
- **resolution** (*int*) – Resolution of the resulting maps. The maps have a equal resolution in kx and ky.
- **out** (*bool*) – Whether or not to export the maps to a .hdf5 file. The file will contain the axis information and is compatible with kMap.
- **out_file** (*str*) – If out is True the path to the output file relative to the root directory of the calculation.

Returns **nd.array** – A (E_num, resolution, resolution) shaped array. First index are the individual slices (even if E_num is 1). Second and third index are kx and ky respectively.

Returns

- **nd.array** – 1D array containing the individual binding energies (not kinetic energy of the e) used for the slices in 'eV'.
- **nd.array** – 1D array containing the kx (and therefore the identical ky) axis values in 1/Angstrom.

`get_gasphase_pes`(*file*, *integrate=('theta', 'phi')*, *plot=False*)

Returns photoemission intensity.

Reads the photoemission intensity from 'file' and, if requested, integrates over 0,1, 2 or all dimensions. Option to plot is only available for integration over 2 dimensions.

Parameters

- **`file`** (*str*) – Relative path from the root directory of the calculation to the PES velocity map file (.vtk file).
- **`integrate`** (*list*) – List containing any combination of 'theta', 'phi' and 'energy' including empty and all of them. The dimensions specified will be integrated over.
- **`plot`** (*bool*) – If len(integrate) != 2 this option will be ignored. Otherwise a basic line plot over the remaining dimension will be returned.

Returns

- **nd.array or float** – The intensity integrated over the dimensions specified in arbitrary units. If integrated over all dimensions this will be a float, otherwise a nd.array with 3 - len(integrate) dimensions. The first dim will be 'energy' or, if integrated over, 'phi'. The last dim will be 'theta' or, if integrated over, 'phi'.
- **tuple or nd.array** – Remaining axes in the correct order. If only one remains this will be a nd.array, if none remains no return value and otherwise a tuple of axes.
- **fig, ax** – If plot == True and integration over two dimensions, the handles for the matplotlib plot.

`get_state_resolved_arpes`(*states=- 1*, *field='probe'*, *omega=None*, *dE_b=0.02*, *resolution=501*, *out=False*, *out_file='output.hdf5'*)

Calculate a state projected (kx, ky) ARPES map for each state.

Using the state projected photoemission intensity files outputed by Octopus (if requested), for each state in 'states' the state projected ARPES map is calculated. The binding energy will be taken from the eigenvalues of the indiviudal states.

Parameters

- **`states`** (*list*) – List of integers specifying the states to be used. For each entry a corresponding file has to be present. If -1 all files found will be used. List can also contain the name of the orbital as string (e.g. 'HOMO-1')
- **`dE_b`** (*float*) – See 'get_gasphase_arpes'.
- **`field`** (*str*) – See 'get_gasphase_arpes'.
- **`omega`** (*float*) – See 'get_gasphase_arpes'.
- **`resolution`** (*int*) – See 'get_gasphase_arpes'
- **`out`** (*bool*) – Whether or not to export the maps to a .hdf5 file. The file will contain the axis information and is compatible with kMap.

- **out_file** (*str*) – If out is True the path to the output file relative to the root directory of the calculation.

Returns

- **nd.array** – A (E_num, resolution, resolution) shaped array. First index are the individual maps (even if len(states) is 1). Second and third index are kx and ky respectively.
- **nd.array** – 1D array containing the individual binding energies (not kinetic energy of the e) used for the states in 'eV'.
- **nd.array** – 1D array containing the kx (and therefore the identical ky) axis values in 1/Angstrom.

1.2.2 DOS

Calculations related to the density of states.

This module contains the 'DOSModel' class, which handles all things related to density of states calculations.

class dos_model.**DOSModel**(*ocotpost*)

DOS model handles all things related to DOS.

Should not be used directly by the user but called by the abstraction layer octopost class.

Parameters **octopost** (*object*) – An instance of the octopost class.

get_atomic_dos(*atom*, *plot=False*, *shift_by_fermi=True*, *plot_fermi=True*, *energy_units='hartree'*, *axes=None*, *combine_orbitals=True*)

Extracts and possibly plots the atom projected density of states.

Returns the density of states projected onto a specific atom.

Parameters

- **atom** (*int*) – Number of the atom the DOS is to be returned.
- **plot** (*bool*) – If True a basic plot will be returned.
- **shift_by_fermi** (*bool*) – If True the energy will be shifted by the fermi energy (automatically extracted) for both the plot and the returned values.
- **plot_fermi** (*bool*) – If True (and plot == True) a dashed line will be plotted as well where the fermi energy is.
- **energy_units** (*str*) – Energy units to be plotted and returned. Options: 'hartree' (= Hartree), 'eV' (= Electron Volt)
- **axes** (*fig, ax*) – Matplotlib handles of a already existing plot to overlay the information of this onto it.
- **combine_orbitals** (*True*) – If True all the orbital projections will be summed up to a single density of states for this atom.

Returns

- **nd.array** – 2D (if combine_orbitals) or (#orbitals of atom + 1)D array containing the energy in the units chosen in the first column and the density of states projected onto the orbitals or the entire atom in the remaining columns.
- **fig, ax** – If plot == True the matplotlib handles for the plot will be returned as well.

get_band_dos(*n_min=1*, *n_max=- 1*, *plot=False*, *shift_by_fermi=True*, *combined=False*, *plot_fermi=True*, *energy_units='hartree'*, *axes=None*)

Extracts and possibly plots the band projected density of states.

Reads the band specific dos files in 'static', if available and returns the total densities of states.

Parameters

- **n_min** (*int*) – The lowest band number to be extracted.
- **n_max** (*int*) – The highest band number to be extracted. -1 means all bands larger n_min in the directory will be used.
- **plot** (*bool*) – If True a basic plot will be returned.
- **shift_by_fermi** (*bool*) – If True the energy will be shifted by the fermi energy (automatically extracted) for both the plot and the returned values.
- **combined** (*bool*) – Whether or not to combine all selected bands into a single total dos.
- **plot_fermi** (*bool*) – If True (and plot == True) a dashed line will be plotted as well where the fermi energy is.
- **energy_units** (*str*) – Energy units to be plotted and returned. Options: 'hartree' (= Hartree), 'eV' (= Electron Volt)
- **axes** (*fig, ax*) – Matplotlib handles of a already existing plot to overlay the information of this onto it.

Returns

- **nd.array** – (n_max - n_min + 1)D array containing the energy in the units chosen in the first column and the density of states projected onto state i in the ith column.
- **fig, ax** – If plot == True the matplotlib handles for the plot will be returned as well.

get_orbital_dos(*orbital*, *plot=False*, *shift_by_fermi=True*, *plot_fermi=True*, *energy_units='hartree'*, *axes=None*, *combine_atoms=True*)

Extracts and possibly plots the orbital projected density of states.

Returns and plots all the densities of states (possibly) combined projected onto a secific atom orbital.

Parameters

- **orbital** (*str*) – Name of the atom orbital to be extracted in the form: 'element symbol' + 'energy level' + 'orbital type'. For example: Cu2p
- **plot** (*bool*) – If True a basic plot will be returned.
- **shift_by_fermi** (*bool*) – If True the energy will be shifted by the fermi energy (automatically extracted) for both the plot and the returned values.
- **plot_fermi** (*bool*) – If True (and plot == True) a dashed line will be plotted as well where the fermi energy is.
- **energy_units** (*str*) – Energy units to be plotted and returned. Options: 'hartree' (= Hartree), 'eV' (= Electron Volt)
- **axes** (*fig, ax*) – Matplotlib handles of a already existing plot to overlay the information of this onto it.
- **combine_atoms** (*bool*) – If true all atoms with such a orbital will be summed up to a single density of states.

Returns

- **nd.array** – 2D (if combine_atoms) or (#orbitals of atom + 1)D array containing the energy in the units chosen in the first column and the density of states projected onto the orbitals or the entire atom in the remaining columns.
- **fig, ax** – If plot == True the matplotlib handles for the plot will be returned as well.

`get_total_dos`(*plot=False*, *shift_by_fermi=True*, *plot_fermi=True*, *energy_units='hartree'*)
Extracts and possibly plots the total density of states.

Reads the 'static/total-dos.dat' file, if available and returns the total density of states.

Parameters

- **plot** (*bool*) – If True a basic plot will be returned.
- **shift_by_fermi** (*bool*) – If True the energy will be shifted by the fermi energy (automatically extracted) for both the plot and the returned values.
- **plot_fermi** (*bool*) – If True (and plot == True) a dashed line will be plotted as well where the fermi energy is.
- **energy_units** (*str*) – Energy units to be plotted and returned. Options: 'hartree' (= Hartree), 'eV' (= Electron Volt)

Returns

- **nd.array** – 2D array containing the enery in the units chosen in the first column and the density of states in the second.
- **fig, ax** – If plot == True the matplotlib handles for the plot will be returned as well.

1.2.3 Bandstructure

Calculations related to the bandstructure.

This module contains the 'BandstructureModel' class, which handles all things related to the bandstructure.

`class` bandstructure_model.`BandstructureModel`(*ocotpost*)
Bandstructure model handles all things related to the bandstructure.

Should not be used directly by the user but called by the abstraction layer octopost class.

Parameters **octopost** (*object*) – An instance of the octopost class.

`get_bandstructure`(*bands=None*, *plot=False*, *energy_units='eV'*)
Returns the bandstructure.

Reads the 'bandstructure' file found in 'static/' and returns the path along the k points as coordinate and (kx, ky, kz) as well as the bands. Optionally a pre-made standard plot can be returned as well.

Parameters

- **bands** (*list*) – List of band indices to be returned. Pass None for all bands.
- **plot** (*bool*) – Returns matplotlib handles for a basic plot of the bandstructure.
- **energy_units** (*str*) – Energy units to be plotted and returned. Options: 'hartree' (= Hartree), 'eV' (= Electron Volt)

Returns

- **nd.array** – Numpy array with all k-points. First column is the coordinate, remaining columns are kx, ky, kz.
- **nd.array** – Numpy array with each column being one band.

- **fig, ax** – Matplotlib handles for the plot. Only returned if plot==True.

1.2.4 Info

Convienence class to extract various data from Octopus output.

class output_read_model.**OutputReadModel**(*ocotpost*)

OutputRead model handles all things related to basic information output.

Should not be used directly by the user but called by the abstraction layer octopost class.

Parameters **octopost** (*object*) – An instance of the octopost class.

get_cell()

Returns lattice vectors.

Returns **nd.array** – 3x3 array where first index runs through the vectors and second through their coordinate.

get_convergence(*plot=False*, *values=('energy', 'density')*)

Returns the convergence parameter energy and density.

Parameters

- **plot** (*bool*) – If True a basic convergence plot for energy and density will be returned.
- **values** (*list*) – List containing strings denoting which values should be examined for convergence. Any combination of: 'energy', 'density'

Returns

- **nd.array** – First column is the SCF cycle. Then energy, energy difference, density and relative density in this order.
- **fig, ax, ax** – Matplotlib handles for the figure, the left axis and the right axis in this order.

get_eigenvalues(*n_max=None*, *n_min=1*, *k_point=None*, *energy_units='hartree'*)

Returns the eigenvalues found in the 'info' file.

Parameters

- **n_max** (*int*) – How many eigenvalues to be returned. If 'None' all of them are returned. If n_max > available eigenvalues all available ones are returned.
- **k_points** (*int*) – At which k_point the eigenvalues are to be read.
- **energy_units** (*str*) – Specify the units in which the energy is returned. Options: 'hartree', 'eV'

Returns **list** – List of eigenvalues (float).

get_fermi(*energy_units='hartree'*)

Returns fermi energy found in the 'total-dos-efermi.dat' file.

Parameters **energy_units** (*str*) – Specify the units in which the energy is returned. Options: 'hartree', 'eV'

Returns **float** – Fermi energy in the specified units.

get_first_brillouin()

Returns the first Brillouin zone in the xy plane.

Calculates the vertices and order of connection between the vertices for the first Brillouin zone in the xy plane.

Returns

- **nd.array** – Nx2 array where first index runs through the vertices of the Brillouin zone (a prior not clear how many) and the second index are the coordinates x and y.
- **list** – Indicies of vertices in order of connection, i.e. vertex number list[0] (in the vertex array) is connected to vertex number list[1] and list[-1].

`get_orbital_state_number`(*state='HOMO'*)
Returns state number of the chosen state. Only works for non periodic systems. Mostly for other Octopost methods.

Parameters **state** (*str*) – State for which the number is to be found. Options: 'homo', 'lumo', 'homo-x', 'lumo+x' where 'x' is any integer > 0

Returns **int** – State number.

`get_reciprocal_cell`()
Returns reciprocal lattice vectors.

Returns **nd.array** – 3x3 array where first index runs through the vectors and second through their coordinate.

`plot_reciprocal`(*unit_vectors=True*, *brillouin=True*, *origin=False*, *axes=None*)
Plots the unit cell, origin and first Brillouin zone in 2D.

Parameters

- **unit_vectors** (*bool*) – Whether or not to plot the reciprocal unit vectors.
- **brillouin** (*bool*) – Whether or not to plot the first Brillouin zone.
- **origin** (*bool*) – Whether or not to plot the origin.
- **axes** – Matplotlib handles for the figure the plots are to be added.

1.2.5 Input

Parser for the input of the Octopus calculation.

A small text parser specifically for the 'parser.log' file of a Octopus calculation containing the inputs done by the user. Used for automatic extraction of certain values necessary for the data post processing.

`class input_parser.Parser`(*ocotpost*)
Textparser class.

Parameters **octopost** (*object*) – An instance of the octopost class.

`input`
A dictionary containing all parsed variables from the parser.log file. Public to directly access not yet abstracted variables.

Type dict

`get_external_field`(*field*)
Returns the external filed specified in the Octopus calculation.

Parameters **field** (*str*) – Each field in a Octopus calculation is given a name. This name is used to find the field data.

Returns **dict** – All available data to this field in form of a dictionary. The available keys depend on the type of field. For more information please see the Octopus documentation: https://octopus-code.org/doc/11.1/html/vars.php?page=alpha

1.2.6 Units

Definition of various useful units used for conversion.

Source for all values unless specified otherwise: https://physics.nist.gov/cuu/Constants/index.html

units.**bohr**
Bohr atomic radius in [Angstrom].

Type float

units.**hartree**
Hartree in [eV].

Type float

units.**m_e**
Electron mass in [kg].

Type float

units.**e**
Elementary charge in [Coulomb].

Type float

units.**energy**(*data*, *from_='hartree'*, *to='eV'*)
Converts energy units.

Parameters

- **data** (*float or list*) – The initual value to be converted in units 'from_'.
- **from_** (*str*) – The unit of the values supplied. Available options are: 'hartree' (= Hartree), 'eV' (= Electron Volt).
- **to** (*str*) – The unit of the output. Available options are: 'hartree', 'eV'.

Returns **float or list** – The data but converted from 'from_' to 'to'.

units.**inverse_length**(*data*, *from_='bohr'*, *to='A'*)
Converts inverse length (reciprocal distance).

Parameters

- **data** (*float or list*) – The initual value to be converted in units 'from_'.
- **from_** (*str*) – The unit of the values supplied. Available options are: 'bohr' (= 1/Bohr radius), 'A' (= 1/Angstrom).
- **to** (*str*) – The unit of the output. Available options are: 'bohr' (= 1/Bohr radius), 'A' (= 1/Angstrom).

Returns **float or list** – The data but converted from 'from_' to 'to'.

1.2.7 Library

Definition of various useful code snipets.

library.**E_to_k**(*E*)

Converts (kinetic) energy (of an electron) to momentum:

```
k = m * v = m * sqrt(2 * E / m) = sqrt(2 * E * m)
```

Parameters **E** (*nd.array or float*) – Energy to be converted in atomic units.

Returns **nd.array or float** – Momentum in atomic units.

library.**cartesian_to_spherical**(*x*, *y*, *z*)

Converts cartesian coordinates to spherical ones:

```
r = sqrt(x^2 + y^2 + z^2)
theta = arccos(z / r)
phi = arctan(y / x)
```

Parameters

- **x** (*nd.array or float*) – x-Coordinate.
- **y** (*nd.array or float*) – y-Coordinate.
- **z** (*nd.array or float*) – z-Coordinate.

Returns

- **nd.array or float** – Radius.
- **nd.array or float** – Polar angle theta.
- **nd.array or float** – Azimuthal angle phi.

library.**find_nearest_index**(*array*, *value*)

Finds the index of the closes element to 'value'.

Parameters

- **array** (*nd.array*) – Array of values.
- **value** (*float*) – Value to be located inside the array.

Returns **int** – Index of the element inside 'array' that is closest to 'value'.

library.**k_to_E**(*k*)

Converts momentum to (kinetic) energy of an electron:

```
E = m * v^2 / 2 = m * (k / m)^2 / 2 = k^2 / (2 * m)
```

Parameters **k** (*nd.array or float*) – Momentum to be converted in atomic units.

Returns **nd.array or float** – Energy in atomic units.

library.**read_vtk_file**(*path*, *array_name*)

Reads a .vtk file containing a structured grid.

Parameters

- **path** (*str*) – Absolute path to the file that is to be read.
- **array_name** (*str*) – Name of the array inside the file to be read.

Returns

- **nd.array** – 4D array. Last dimension holds length 3 tuple with x, y and z coordinate for that point.
- **nd.array** – 3D array with data at each point.

PYTHON MODULE INDEX

INDEX

L

M

O

P

R

U