Hylight

Release 1.0.0

Théo Cavignac

Feb 10, 2023

CONTENTS

CHAPTER

ONE

ABOUT

Welcome to Hylight's documentation!

Hylight is a post-processing tool written in Python to simulate the luminescence of solids based on the results of *ab initio* computations.

You may be interested in reading our first paper using it on the luminescence of $BaZrO_3$:Ti [BaZrO3:Ti].

To learn how to use Hylight, you can read the tutorial. Finally there is a complete reference that goes over all individual functions implemented in the package.

CHAPTER

TWO

ATTRIBUTION

2.1 Licenses

Hylight is written and maintained by the PyDEF team. The source code for Hylight is distributed under the GPLv3 license. This documentation is distributed under CC-BY license.

2.2 Citing Hylight

If you ever use Hylight for a published scientific work, we would ask you to cite the related paper [Hylight].

CHAPTER

THREE

CONTENT

3.1 BaZrO3:Ti study

Here are some utility functions and import that will be used throughout the study.

```
%matplotlib inline
```

```
import numpy as np
import matplotlib.pyplot as plt
from hylight.constants import *

eV1_in_nm = h_c / eV_in_J * 1e9
```

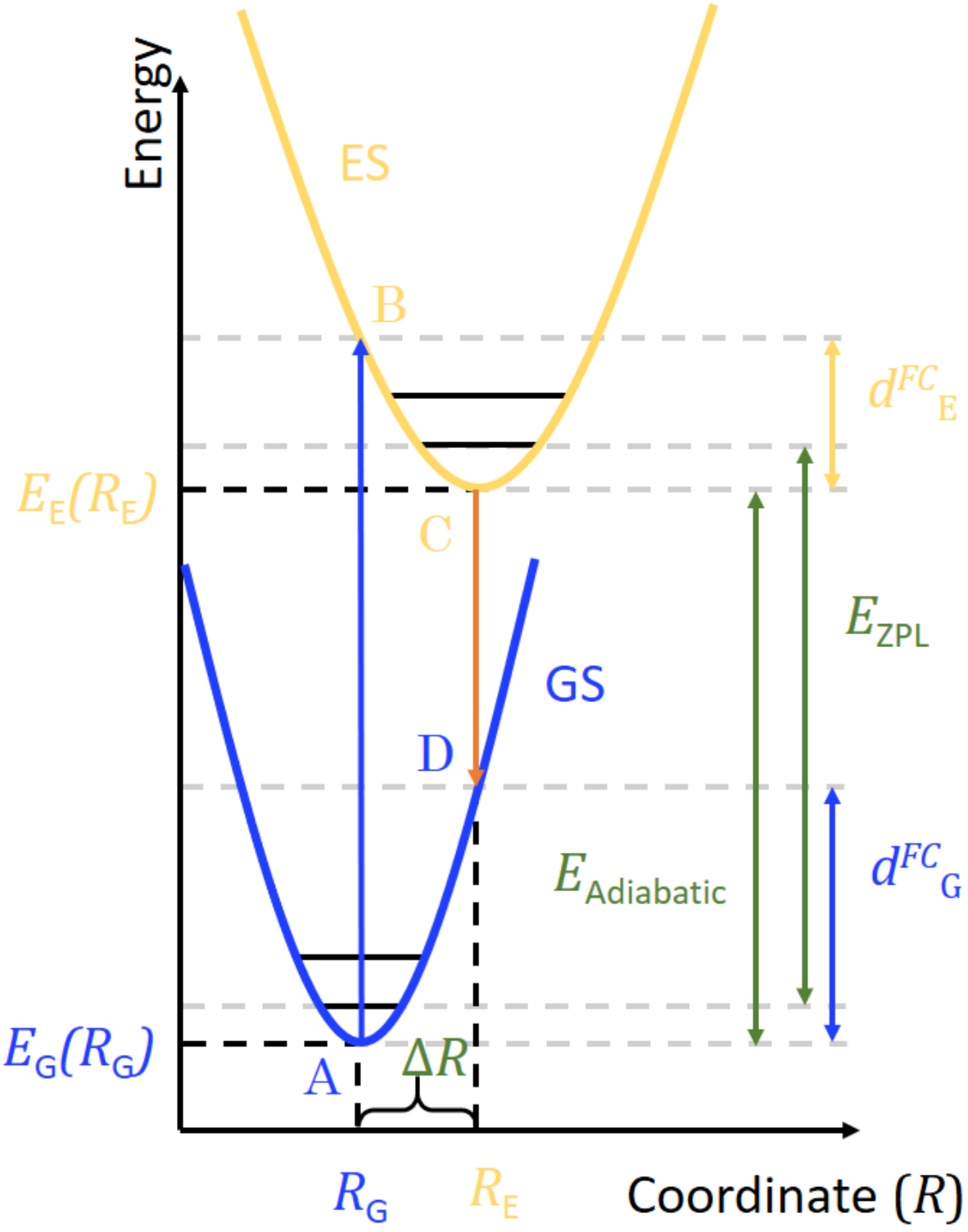
```
def best_max(raw_x, raw_y, f=0.95):
    "Fit a 2nd order polynom to get a position of the maximum without the noise."
    guess = np.max(raw_y)

    y = raw_y[raw_y > f * guess]
    x = raw_x[raw_y > f * guess]

    x1, x2, x3, x4 = np.mean(x**1), np.mean(x**2), np.mean(x**3), np.mean(x**4)

    x2y = np.mean(x**2 * y)
    xy = np.mean(x * y)
    my = np.mean(y)

    A = np.array([[x4, x3, x2],
                  [x3, x2, x1],
                  [x2, x1, 1.]])

    B = np.array([[x2y, xy, my]]).transpose()

    alpha, beta, gamma = np.linalg.solve(A, B)[:, 0]

    return -0.5 * beta / alpha, gamma - 0.25 * beta**2 / alpha

def load_exp(path, sep=";", skip=0):
    exp = np.loadtxt(path,
                     delimiter=sep,
                     skiprows=skip)
```

(continues on next page)

(continued from previous page)

```
    x, y = eV1_in_nm / exp[:, 0], exp[:, 1]
    xmax, ymax = best_max(x, y)
    y /= ymax
    return x, y, xmax

def select_interval(x, y, emin, emax, norm=True, npoints=None):
    slice_ = (x > emin) * (x < emax)
    xs, ys = x[slice_], y[slice_] / (np.max(y[slice_]) if norm else 1.)

    if npoints is not None:
        emin = max(np.min(xs), emin)
        emax = min(np.max(xs), emax)
        xint = np.linspace(emin, emax, npoints)
        return xint, interp1d(xs, ys)(xint)

    return xs, ys
```

To perform the simulation of the spectrum of BaZrO3:Ti we need a few pieces of information. The following diagram shows the four points A, B, C and D that need to be computed in DFT. We used constrained DFT to compute B and C with an explicit hole in the VBM and an electron in the CBM. The cDFT and the regular DFT computations provides us with total energies and the positions R_G and R_E that we will use as inputs in the next section.



Given the geometry R_G we also perform a DFPT computation to get the eigen values and eigenvectors of the dynamical matrix, that is we get the Γ vibrational modes of the crystal.

The modes are extracted from the VASP output file and stored in the `OUTCAR.npz` with the command line tool `hylight-modes`:

```
$ hylight-modes vasp OUTCAR OUTCAR.npz
Loaded 120 modes from OUTCAR.
Wrote OUTCAR.npz.
```

3.1.1 Vibronic luminescence spectrum

Electronic and vibrational parameters

Here we declare the inputs that will be used later.

```
fc_shift_es = 0.1091  # B-C
fc_shift_gs = 0.2871  # D-A
delta_e_em = 2.6905  # C-D

e_adia = delta_e_em + fc_shift_gs  # C-A

outcar = "OUTCAR.npz"  # vibrations
poscar_gs = "POSCAR"  # R_G
poscar_es = "POSCAR_ES"  # R_E
T = 300  # measure temperature (K)

print(f"Adiabatic energy difference: {e_adia:.3} eV")
```

```
Adiabatic energy difference: 2.98 eV
```

Simulating the spectrum

Here we load the experimental data that will be used for comparision with the simulation.

```
exp_e, exp_i, max_emission = load_exp("emission_300K.csv", skip=1)

print(f"Measured maximum of emission: {max_emission:.3} eV")
```

```
Measured maximum of emission: 2.73 eV
```

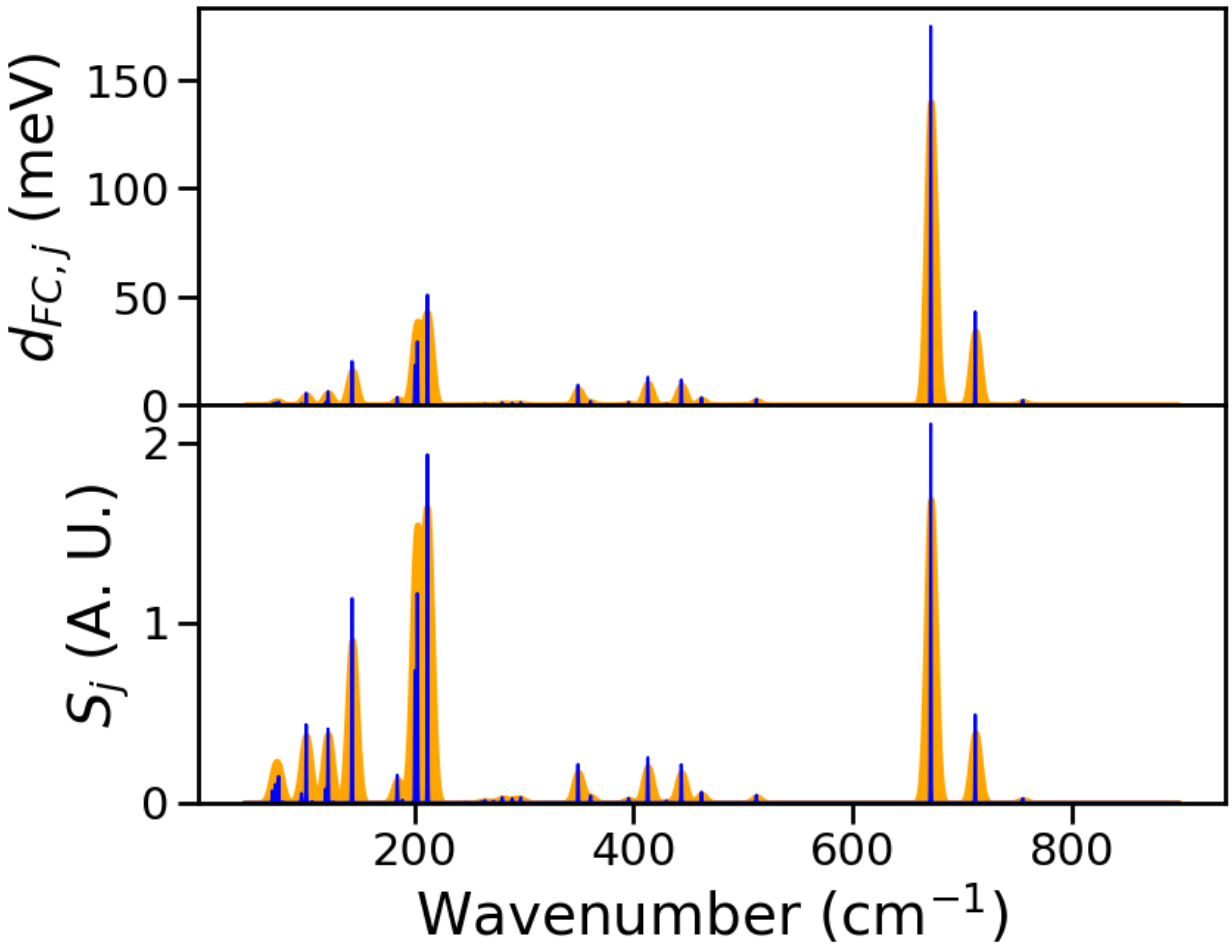
We now plot the spectral function to identify important modes.

```
from hylight.multi_phonons import plot_spectral_function

fig, (ax_fc, ax_s) = plot_spectral_function(outcar, poscar_gs, poscar_es, use_cm1=True,␣
↪disp=5e-1, mpl_params={
    "S_stack": {"color": "orange"},
    "FC_stack": {"color": "orange"},
    "S_peaks": {"color": "blue", "lw": 1.5},
    "FC_peaks": {"color": "blue", "lw": 1.5},
})

ax_fc.set_ylabel("$d_{FC,j}$ (meV)")
```

(continues on next page)

(continued from previous page)

```
ax_s.set_ylabel("$S_j$ (A. U.)")

fig.set_size_inches((9, 7))
```



We simulate the spectrum using the Mode Independant scaling approximation for the ES PES.

```
from hylight.multi_phonons import spectrum, ExPES, OmegaEff
import logging

logging.getLogger().setLevel(logging.INFO)  # setting the logging level to INFO

e, sp = spectrum(  # simulate the spectrum
    outcar,
    poscar_gs,
    poscar_es,
    e_adia,
    T,
    fc_shift_gs,
    fc_shift_es,
    ex_pes=ExPES.ISO_SCALE,
    omega_eff_type=OmegaEff.FC_MEAN,
```

(continues on next page)

(continued from previous page)

```
)

sp /= np.max(sp)  # normalize the spectrum

_, max_th = max(zip(sp, e))  # extract the maximum of emission energy
print(f"Computed maximum of emission: {max_th:.3} eV")
```

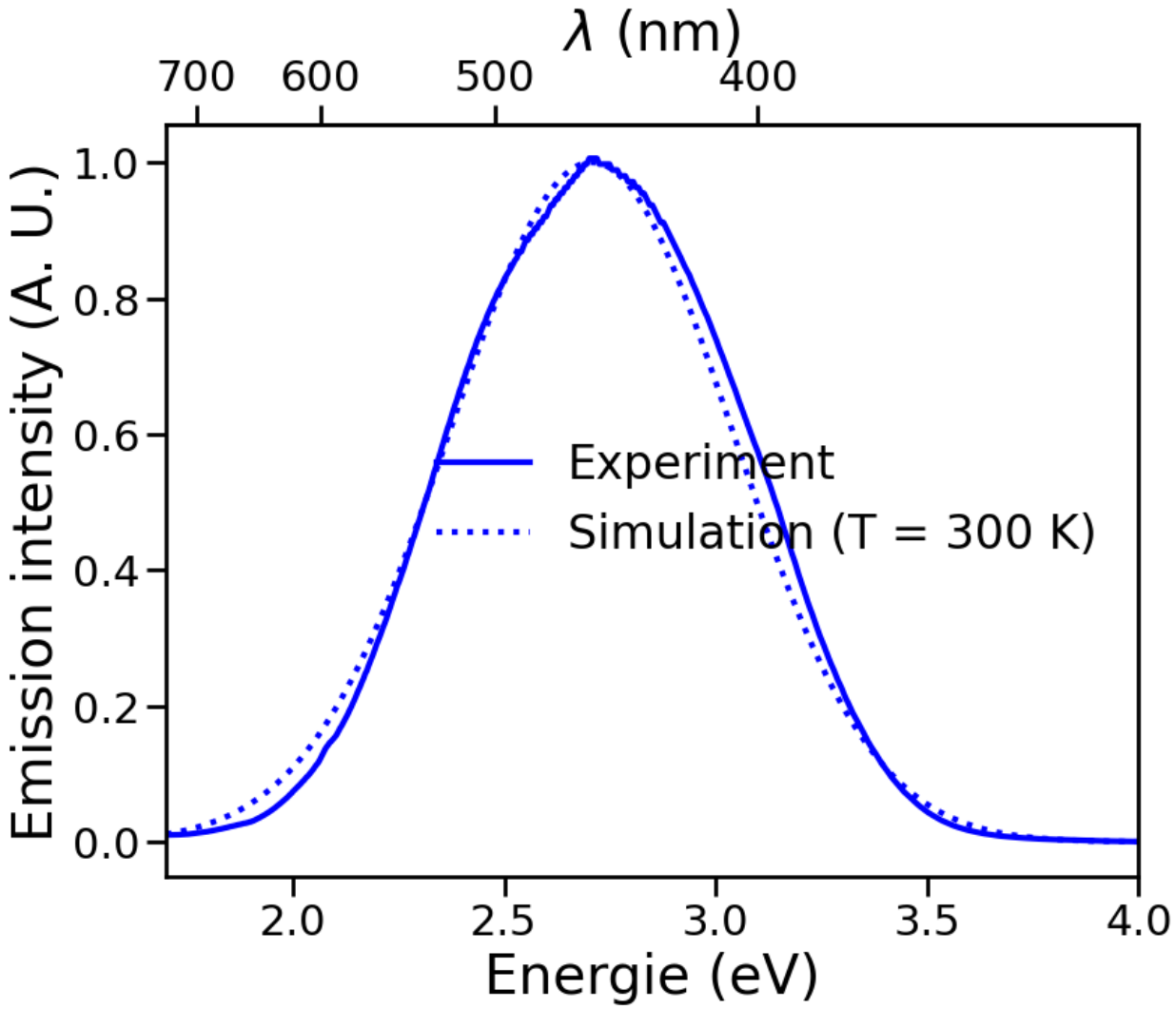
```
d_fc^e,v = 0.15494048765687785
omega_gs = 59.3441936353856 meV 478.64320119276204 cm-1
omega_es = 36.58255313775182 meV 295.0581896729496 cm-1
S_em = 10.055832452822724
d_fc^g,v = 0.407730650836752
FWHM 723.172263202136 meV
FWHM 0K 564.4130957462941 meV
```

```
Using a Gaussian line shape.
Computed maximum of emission: 2.71 eV
```

Finally we setup a plot to show the result of the simulation against the measurement.

```
plt.figure(figsize=(9, 7))

e, sp = e[1:], sp[1:]
exp_e, exp_i = exp_e[1:], exp_i[1:]
plt.plot(exp_e, exp_i, "b", lw=3, label="Experiment")
plt.plot(e, sp, ":", color="b", lw=3, label=f"Simulation (T = {T} K)")

plt.xlim(1.7, 4)
plt.xlabel("Energie (eV)")
plt.ylabel("Emission intensity (A. U.)")

ax = plt.gca()
np.seterr(divide='ignore')  # there is a division by zero occuring in the next line but␣
↪it is irrelevant for our xlim
secax = ax.secondary_xaxis('top', functions=(lambda x: eV1_in_nm / x, lambda x: eV1_in_
↪nm / x))
secax.set_xlabel("$\lambda$ (nm)")

plt.legend(prop={"size": 24})
```

```
<matplotlib.legend.Legend at 0x7f8e5876e530>
```



3.2 hylight package

3.2.1 Subpackages

hylight.crystal package

Submodules

hylight.crystal.loader module

Read vibrational modes from CRYSTAL log.

`hylight.crystal.loader.`**`load_phonons`**(*path*)

Load phonons from a CRYSTAL17 logfile.

Returns

(phonons, pops, masses) phonons: list of hylight.mode.Mode instances pops: population for each

atom species masses: list of SI masses

hylight.crystal.loader.**normalize**(*name*)

Module contents

hylight.findiff package

Submodules

hylight.findiff.collect module

Grab forces from a collection of single point computations and compute hessian.

hylight.findiff.collect.**dropwhile_err**(*pred*, *it*, *else_err*)

itertools.dropwhile wrapper that raise else_err if it reach the end of the file.

hylight.findiff.collect.**get_forces**(*n*, *path*)

Extract n forces from an OUTCAR.

hylight.findiff.collect.**get_forces_and_pos**(*n*, *path*)

Extract n forces and atomic positions from an OUTCAR.

hylight.findiff.collect.**get_ref_info**(*path*)

Load system infos from a OUTCAR.

This is an ad hoc parser, so it may fail if the OUTCAR changes a lot.

Returns

(atoms, ref, pops, masses) atoms: list of species names pos: positions of atoms pops: population for each atom species masses: list of SI masses

hylight.findiff.collect.**process_phonons**(*outputs*, *ref_output*, *basis_source=None*, *amplitude=0.01*, *nproc=1*, *symm=True*)

Process a set of OUTCAR files to compute some phonons using Force based finite differences.

Parameters

- **outputs** – list of OUTCAR paths corresponding to the finite displacements.
- **ref_output** – path to non displaced OUTCAR.
- **basis_source** – read a displacement basis from a path. The file is a npy file from numpy's save. If None, the basis is built from the displacements. If not None, the order of outputs *must* match the order of the displacements in the array.
- **amplitude** – amplitude of the displacement, only used if basis_source *is not* None.
- **nproc** – number of parallel processes used to load the files.
- **symm** – If True, use symmetric differences. OUTCARs *must* be ordered as [+delta_1, -delta_1, +delta_2, -delta_2, ...].

Returns

the same tuple as the load_phonons functions.

Remark: When using non canonical basis (displacements are not along a single degree of freedom of a single atom at a time) it may be important to provide the basis excplicity because it will avoid important rounding errors found in the OUTCAR.

hylight.findiff.gen module

Generate a collection of positions for finite differences computations.

hylight.findiff.gen.**gen_disp**(*ref*, *basis*, *amplitude=0.01*, *symm=False*)

Iterate over displaced Poscar instances from a given set of directions.

Takes a reference position and displaces it into directions from basis to produce a set of positions for finite differences computations.

Parameters

- **ref** – reference Poscar instancce
- **basis** – nupy array where each row is a direction.
- **amplitude** – (optional, 0.01) amplitude of the displacements in Angstrom.
- **symm** – (optional, False) when True, generate both ref+delta and ref-delta POSCARs for each direction.

hylight.findiff.gen.**random_basis**(*n*, *seed=0*)

Generate a random basis matrix or rank n.

Parameters

- **n** – Rank of the basis
- **seed** – (optional, 0) randomness seed.

Returns

a (n, n) orthonormal numpy array.

hylight.findiff.gen.**save_disp**(*ref*, *basis*, *disp_dest='.'*, *amplitude=0.01*, *symm=False*)

Produces displaced POSCARs from a given set of directions.

Takes a reference position and displaces it into directions from basis to produce a set of positions for finite differences computations.

Parameters

- **ref** – reference Poscar instancce
- **basis** – nupy array where each row is a direction.
- **disp_dest** – (optional, ".") directory name or callable that determine the destination of the POSCARs.
- **amplitude** – (optional, 0.01) amplitude of the displacements in Angstrom.
- **symm** – (optional, False) when True, generate both ref+delta and ref-delta POSCARs for each direction.

hylight.findiff.gen.**save_disp_from_basis**(*source*, *basis_source*, *disp_dest='.'*, *amplitude=0.01*, *symm=False*)

Produces displaced POSCARs from a given set of directions.

Takes a reference position and displaces it into directions from basis_source to produce a set of positions for finite differences computations.

Parameters

- **source** – reference POSCAR

- **basis_source** – Name of the basis set file. It is a npy file made with numpy's save function. Each row is a direction.
- **disp_dest** – (optional, ".") directory name or callable that determine the destination of the POSCARs.
- **amplitude** – (optional, 0.01) amplitude of the displacements in Angstrom.
- **symm** – (optional, False) when True, generate both ref+delta and ref-delta POSCARs for each direction.

hylight.findiff.gen.**save_random_disp**(*source*, *disp_dest='.'*, *basis_dest='basis.npy'*, *amplitude=0.01*, *seed=0*, *symm=False*)

Produces displaced POSCARs in random directions.

Takes a reference position and displaces it into random directions to produce a set of positions for finite differences computations. It also save the set of displacements in basis_dest.

Parameters

- **source** – reference POSCAR
- **disp_dest** – (optional, ".") directory name or callable that determine the destination of the POSCARs.
- **basis_dest** – (optional, "basis.npy") Name of the basis set file. It is a npy file made with numpy's save function.
- **amplitude** – (optional, 0.01) amplitude of the displacements in Angstrom.
- **seed** – (optional, 0) random seed to use to generate the displacements.
- **symm** – (optional, False) when True, generate both ref+delta and ref-delta POSCARs for each direction.

Module contents

A module for finite differences computations.

hylight.jmol package

Module contents

Write Jmol files.

hylight.jmol.**export**(*dest*, *mode*, *compression=8*, ***opts*)

Export a mode to JMol zip format.

Parameters

- **dest** – path to the JMol zip file.
- **mode** – the mode to export.
- **compression** – (optional) zipfile compression algorithm.
- **unitcell** – lattice vectors as a 3x3 matrix where vectors are in rows.
- **bonds** – a list of *(sp_a, sp_b, min_dist, max_dist)* where species names are strings of names of species and **_dist* are interatomic distances in Angstrom.

- **atom_colors** – a list of (sp, color) where sp is the name of a species and color is the name of a color or an HTML hex code (example "#FF0000" for pure red).

hylight.jmol.**export_disp**(*dest*, *struct*, *disp*, *compression=8*, ***opts*)

Export a difference between two positions to JMol zip format.

Parameters

- **dest** – path to the JMol zip file.
- **struct** – the reference position (a *hylight.struct.Struct* instance).
- **disp** – an array of displacements.
- **compression** – (optional) zipfile compression algorithm.
- **unitcell** – lattice vectors as a 3x3 matrix where vectors are in rows.
- **bonds** – a list of *(sp_a, sp_b, min_dist, max_dist)* where species names are strings of names of species and **_dist* are interatomic distances in Angstrom.
- **atom_colors** – a list of (sp, color) where sp is the name of a species and color is the name of a color or an HTML hex code (example "#FF0000" for pure red).

hylight.jmol.**format_v**(*v*)

hylight.jmol.**write_jmol_options**(*f*, *opts*)

Write options in the form of a JMol script.

Parameters

- **f** – a file like object.
- **opts** – a set of options (see export).

hylight.jmol.**write_xyz**(*f*, *atoms*, *ref*, *delta*)

Write the coordinates and displacements of in the JMol xyz format.

Parameters

- **f** – a file-like object to write to.
- **atoms** – the list of atom names
- **ref** – an array of atomic positions
- **delta** – an array of displacements

hylight.phonopy package

Submodules

hylight.phonopy.loader module

Module to load phonons frequencies and eigenvectors from phonopy output files.

It always uses PyYAML, but it can also need h5py to read hdf5 files.

class hylight.phonopy.loader.**Struct**(*pops: 'list[int]'*, *lattice: 'NDArray[float]'*, *masses: 'list[float]'*, *atoms: 'list[str]'*, *ref: 'NDArray[float]'*)

Bases: object

atoms: list[str]

classmethod from_yaml_cell(*cell: dict*) → *Struct*

lattice: ndarray[Any, dtype[float]]

masses: list[float]

pops: list[int]

ref: ndarray[Any, dtype[float]]

hylight.phonopy.loader.**get_struct**(*phyaml: str*) → *Struct*

hylight.phonopy.loader.**load_phonons**(*dir_: str*) → tuple[list[*hylight.mode.Mode*], list[int], list[float]]

hylight.phonopy.loader.**load_phonons_bandsh5**(*bandh5: str*, *phyaml: str*, *op=<built-in function open>*) → tuple[list[*hylight.mode.Mode*], list[int], list[float]]

hylight.phonopy.loader.**load_phonons_bandyaml**(*bandy: str*) → tuple[list[*hylight.mode.Mode*], list[int], list[float]]

hylight.phonopy.loader.**load_phonons_qpointsh5**(*qph5: str*, *phyaml: str*, *op=<built-in function open>*) → tuple[list[*hylight.mode.Mode*], list[int], list[float]]

hylight.phonopy.loader.**load_phonons_qpointsyaml**(*qpyaml: str*, *phyaml: str*) → tuple[list[*hylight.mode.Mode*], list[int], list[float]]

Module contents

hylight.vasp package

Submodules

hylight.vasp.common module

Common utilities to read and write VASP files.

class hylight.vasp.common.**Poscar**(*cell_parameters*, *species*, *species_names=None*)

Bases: *Struct*

classmethod from_file(*filename*)

Load a POSCAR file

Parameters
filename – path to the file to read

Returns
a Poscar object.

to_file(*path='POSCAR'*, *cartesian=True*)

Write a POSCAR file

The property system_name may be set to change the comment at the top of the file.

Parameters

- **path** – path to the file to write

- **cartesian** – if True, write the file in cartesian representation, if False, write in fractional representation

hylight.vasp.loader module

Read vibrational modes from VASP files.

hylight.vasp.loader.**load_phonons**(*path*)

Load phonons from a OUTCAR.

Remark: This function is a bit heavy because of text parsing. You may want to use hyligh-modes to parse one for all the file and later load that preparsed file using npz.load_phonons instead.

Returns

(phonons, pops, masses) phonons: list of hylight.mode.Mode instances pops: population for each atom species masses: list of SI masses

hylight.vasp.loader.**load_poscar**(*path*)

Read the positions from a POSCAR.

Returns

a np.ndarray((natoms, 3), dtype=float)

hylight.vasp.loader.**load_poscar_latt**(*path*)

Read the positions and the lattice parameters from a POSCAR.

Returns

a (np.ndarray((natoms, 3), dtype=float), nd.array((3, 3), dtype=float)) first element is the set of positions second element is the lattice parameters

hylight.vasp.utils module

Pervasive utilities for hylight.vasp submodule.

hylight.vasp.utils.**make_finite_diff_poscar**(*outcar*, *poscar_gs*, *poscar_es*, *A=0.01*, *load_phonons=<function load_phonons>*, *bias=0*, *mask=None*)

Compute positions for evaluation of the curvature of the ES PES.

Parameters

- **outcar** – the name of the file where the phonons will be read.
- **poscar_gs** – the path to the ground state POSCAR
- **poscar_es** – the path to the excited state POSCAR, it will be used as a base for the generated Poscars
- **A** – (optional, 0.01) the amplitude of the displacement in A.
- **load_phonons** – (optional, vasp.loader.load_phonons) the procedure use to read outcar

Returns

(mu, pes_left, pes_right) mu: the effective mass in kg pes_left: a Poscar instance representing the left displacement pes_right: a Poscar instance representing the right displacement

Module contents

3.2.2 Submodules

3.2.3 hylight.constants module

A collection of useful physics constants.

3.2.4 hylight.loader module

Wrapper for the npz loader.

3.2.5 hylight.mode module

Vibrational mode and related utilities.

class hylight.mode.**Mode**(*atoms*, *n*, *real*, *energy*, *ref*, *eigenvector*, *masses*)

Bases: object

The representation of a vibrational mode.

It stores the eigenvector and eigendisplacement. It can be used to project other displacement on the eigenvector.

huang_rhys(*delta_R*)

Compute the Huang-Rhyes factor

$$S_i = 1/2\frac{\omega}{\hbar}[(M^{1/2^T}\Delta R)\cdot\gamma_i]^2 = 1/2\frac{\omega}{\hbar}\sum_i m_i^{1/2}\gamma_i\Delta R_i{}^2$$

Parameters
delta_R – displacement in SI

project(*delta_Q*)

Project delta_Q onto the eigenvector

project_coef2(*delta_Q*)

Square lenght of the projection of delta_Q onto the eigenvector.

project_coef2_R(*delta_R*)

Square lenght of the projection of delta_R onto the eigenvector.

to_jmol(***opts*)

Write a mode into a Jmol file.

See *hylight.jmol.export()* for the parameters.

to_traj(*duration*, *amplitude*, *framerate=25*)

Produce a ase trajectory for animation purpose.

Parameters

- **duration** – duration of the animation in seconds
- **amplitude** – amplitude applied to the mode in A (the modes are normalized)
- **framerate** – number of frame per second of animation

hylight.mode.**dynamical_matrix**(*phonons*)

Retrieve the dynamical matrix from a set of modes.

Note that if some modes are missing the computation will fail.

Parameters
phonons – list of modes

hylight.mode.**get_HR_factors**(*phonons*, *delta_R_tot*, *mask=None*)

Compute the Huang-Rhys factors for all the real modes with energy above bias.

Parameters

- **phonons** – list of modes
- **delta_R_tot** – displacement in SI
- **mask** – a mask to filter modes based on their energies.

hylight.mode.**get_energies**(*phonons*, *mask=None*)

Return an array of mode energies in SI

hylight.mode.**rot_c_to_v**(*phonons*)

Rotation matrix from Cartesian basis to Vibrational basis (right side).

3.2.6 hylight.mono_phonon module

Simulation of spectra in 1D model.

hylight.mono_phonon.**compute_spectrum**(*e*, *e_zpl*, *S*, *sig*, *e_phonon_g*)

Compute a spectrum from 1D model with experimental like inputs

Parameters

- **e** – a numpy array of energies to compute the spectrum at
- **e_zpl** – energy of the zero phonon line, in eV
- **S** – the emission Huang-Rhys factor
- **sig** – the lineshape standard deviation

E_phonon_g
the ground state phonon energy

hylight.mono_phonon.**huang_rhys**(*stokes_shift*, *e_phonon*)

Huang-Rhys factor from the Stokes shift and the phonon energy.

hylight.mono_phonon.**sigma**(*T*, *S*, *e_phonon*)

hylight.mono_phonon.**sigma_soft**(*T*, *S_em*, *e_phonon_g*, *e_phonon_e*)

Temperature dependant standard deviation of a line dependant.

Parameters

- **T** – temperature in K
- **S_em** – emmission Huang-Rhys factor
- **e_phonon_g** – energy of the GS PES vibration (eV)
- **e_phonon_e** – energy of the ES PES vibration (eV)

hylight.mono_phonon.**spectrum**(*e_zpl*, *T*, *fc_shift_g*, *fc_shift_e*, *e_phonon_g*, *hard_osc=False*, *n_points=5000*, *e_min=0*, *e_max=5*)

Compute a spectrum from a single vibrational mode and some energetic terms.

Parameters

- **e_zpl** – energy of the zero phonon line, in eV
- **T** – temperature in K
- **fc_shift_g** – Franck-Condon shift of emmission in eV
- **fc_shift_e** – Franck-Condon shift of absobtion in eV
- **e_phonon_g** – Mode energy in ground state in eV
- **hard_osc** – boolean, use the hard oscillator mode: vibration mode has the same energy in GD and ES
- **n_points** – number of points in the spectrum
- **e_min** – energy lower bound for the spectrum, in eV
- **e_max** – energy higher bound for the spectrum, in eV

3.2.7 hylight.multi_phonons module

Simulation of spectra in nD model.

class hylight.multi_phonons.**ExPES**(*wm*)

Bases: object

Mode of approximation of the ES PES curvature.

ISO_SCALE: We suppose eigenvectors are the same, but the frequencies are scaled by a constant factor.

SINGLE_ES_FREQ: We suppose all modes have the same frequency (that should be provided).

FULL_ND: (Not implemented) We know the frequency of each ES mode.

FULL_ND: *ExPES* = <hylight.multi_phonons.ExPES object>

ISO_SCALE: *ExPES* = <hylight.multi_phonons.ExPES object>

SINGLE_ES_FREQ: *ExPES* = <hylight.multi_phonons.ExPES object>

class hylight.multi_phonons.**LineShape**(*value*)

Bases: Enum

Line shape type.

GAUSSIAN = 0

LORENTZIAN = 1

NONE = 2

class hylight.multi_phonons.**Mask**(*intervals*)

Bases: object

accept(*value*)

add_interval(*interval*)

as_bool(*ener*)

classmethod from_bias(*bias*)

plot(*ax*, *unit*)

reject(*value*)

class hylight.multi_phonons.**OmegaEff**(*value*)

Bases: Enum

Mode of computation of a effective frequency.

FC_MEAN:

$$\Omega = \frac{\sum_j \omega_j d_j^{\text{FC}}}{\sum_j d_j^{\text{FC}}}$$

Should be used with *ExPES.ISO_SCALE* because it is associated with the idea that all the directions are softened equally in the excited state.

HR_MEAN:

$$\Omega = \frac{d^{\text{FC}}}{S_{\text{tot}}} = \frac{\sum_j \omega_j S_j}{\sum_j S_j}$$

HR_RMS:

$$\Omega = \sqrt{\frac{\sum_j \omega_j^2 S_j}{\sum_j S_j}}$$

FC_RMS:

$$\Omega = \sqrt{\frac{\sum_j \omega_j^2 d_j^{\text{FC}}}{\sum_j d_j^{\text{FC}}}}$$

Should be used with *ExPES.SINGLE_ES_FREQ* because it makes sense when we get only one Omega_eff for the excited state (this single effective frequency should be computed beforehand)

ONED_FREQ:

$$\Omega = \frac{(\Delta Q^T D \Delta Q)}{\Delta Q^2}$$

Correspond to the actual curvature in the direction of the displacement.

FC_MEAN = 0

FC_RMS = 3

HR_MEAN = 1

HR_RMS = 2

ONED_FREQ = 4

hylight.multi_phonons.**compute_delta_R**(*poscar_gs*, *poscar_es*)

Return $Delta R$ in A.

Parameters

- **poscar_gs** – path to ground state positions file.
- **poscar_es** – path to excited state positions file.

Returns

a np.array of (n, 3) shape where n is the number of atoms.

hylight.multi_phonons.**compute_spectrum**(*phonons*, *delta_R*, *zpl*, *fwhm*, *e_max*, *resolution_e*, *bias=0*, *mask=None*, *window_fn=<function hamming>*, *pre_convolve=None*, *shape=LineShape.GAUSSIAN*)

Compute a luminescence spectrum with the time-dependant formulation with an arbitrary linewidth.

Parameters

- **phonons** – list of modes
- **delta_R** – displacement in A
- **zpl** – zero phonon line energy in eV
- **fwhm** – zpl lineshape full width at half maximum in eV or None if fwhm is None: the raw spectrum is not convoluted with a line shape if fwhm < 0: the spectrum is convoluted with a lorentizan line shape if fwhm > 0: the spectrum is convoluted with a gaussian line shape
- **e_max** – max energy in eV (should be at least > 2*zpl)
- **resolution_e** – energy resolution in eV
- **bias** – ignore low energy vibrations under bias in eV
- **window_fn** – windowing function in the form provided by numpy (see numpy.hamming)
- **pre_convolve** – (float, optional, None) if not None, standard deviation of the pre convolution gaussian

Returns

(energy_array, intensity_array)

hylight.multi_phonons.**compute_spectrum_soft**(*phonons*, *delta_R*, *zpl*, *T*, *fc_shift_gs*, *fc_shift_es*, *e_max*, *resolution_e*, *mask=None*, *window_fn=<function hamming>*, *pre_convolve=None*, *shape=LineShape.GAUSSIAN*, *omega_eff_type=OmegaEff.FC_MEAN*, *result_store=None*, *ex_pes=<hylight.multi_phonons.ExPES object>*, *correct_zpe=False*)

Compute a spectrum without free parameters.

Parameters

- **phonons** – list of modes
- **delta_R** – displacement in A
- **zpl** – zero phonon line energy in eV
- **T** – temperature in K
- **fc_shift_gs** – Ground state/absorption Franck-Condon shift in eV
- **fc_shift_es** – Exceited state/emmission Franck-Condon shift in eV

- **e_max** – max energy in eV (should be at least > 2*zpl)
- **resolution_e** – energy resolution in eV
- **bias** – ignore low energy vibrations under bias in eV
- **window_fn** – windowing function in the form provided by numpy (see numpy.hamming)
- **pre_convolve** – (float, optional, None) if not None, standard deviation of the pre convolution gaussian
- **shape** – ZPL line shape.
- **omega_eff_type** – mode of evaluation of effective frequency.
- **result_store** – a dictionary to store some intermediate results.
- **ex_pes** – mode of evaluation of the ES PES curvature.
- **correct_zpe** – correct the ZPL to take the zero point energy into account.

Returns
(energy_array, intensity_array)

hylight.multi_phonons.**compute_spectrum_width_ah**(*phonons_gs*, *phonons_es*, *delta_R*, *zpl*, *T*, *e_max*, *resolution_e*, *bias=0*, *mask=None*, *window_fn=<function hamming>*, *pre_convolve=None*, *shape=LineShape.GAUSSIAN*, *result_store=None*, *correct_zpe=False*)

Luminescence spectrum with a line width that takes the ES PES curvature into account.

Uses the full Dushinsky matrix for the computation of the line width, but assume the Dushinsky matrix to be the identity in the computation of the FC integrals (thus ignoring mode mixing).

Parameters

- **phonons** – list of modes
- **delta_R** – displacement in A
- **zpl** – zero phonon line energy in eV
- **T** – temperature in K
- **fc_shift_gs** – Ground state/absorption Franck-Condon shift in eV
- **fc_shift_es** – Exceited state/emmission Franck-Condon shift in eV
- **e_max** – max energy in eV (should be at least > 2*zpl)
- **resolution_e** – energy resolution in eV
- **bias** – ignore low energy vibrations under bias in eV
- **window_fn** – windowing function in the form provided by numpy (see numpy.hamming)
- **pre_convolve** – (float, optional, None) if not None, standard deviation of the pre convolution gaussian
- **shape** – ZPL line shape.
- **omega_eff_type** – mode of evaluation of effective frequency.
- **result_store** – a dictionary to store some intermediate results.
- **ex_pes** – mode of evaluation of the ES PES curvature.
- **correct_zpe** – correct the ZPL to take the zero point energy into account.

Returns
(energy_array, intensity_array)

hylight.multi_phonons.**duschinsky**(*phonons_a*, *phonons_b*)

Dushinsky matrix from b to a $S_{a \gets b}$.

hylight.multi_phonons.**dynmatshow**(*dynmat*, *blocks=None*)

Plot the dynamical matrix.

Parameters

- **dynmat** – numpy array representing the dynamical matrice in SI.
- **blocks** – (optional) a list of coloured blocks in the form *(label, number_of_atoms, color)*.

hylight.multi_phonons.**effective_phonon_energy**(*omega_eff_type*, *hrs*, *es*, *masses*, *delta_R=None*)

Compute an effective phonon energy in eV following the strategy of omega_eff_type.

Parameters

- **omega_eff_type** – The mode of evaluation of the effective phonon energy.
- **hrs** – The array of Huang-Rhyes factor for each mode.
- **es** – The array of phonon energy in eV.
- **masses** – The array of atomic masses in atomic mass unit.
- **delta_R** – The displacement between GS and ES in A. It is only required if omega_eff_type is ONED_FREQ.

Returns
The effective energy in eV.

hylight.multi_phonons.**fc_spectrum**(*phonons*, *delta_R*, *n_points=5000*, *disp=1*)

hylight.multi_phonons.**freq_from_finite_diff**(*left*, *mid*, *right*, *mu*, *A=0.01*)

Compute a vibration energy from three energy points.

Parameters

- **left** – energy (eV) of the left point
- **mid** – energy (eV) of the middle point
- **right** – energy (eV) of the right point
- **mu** – effective mass associated with the displacement from the middle point to the sides.
- **A** – amplitude (A) of the displacement between the middle point and the sides.

hylight.multi_phonons.**hr_spectrum**(*phonons*, *delta_R*, *n_points=5000*, *disp=1*)

hylight.multi_phonons.**plot_spectral_function**(*mode_source*, *poscar_gs*, *poscar_es*, *load_phonons=<function load_phonons>*, *use_cm1=False*, *disp=1*, *mpl_params=None*, *mask=None*)

Plot a two panel representation of the spectral function of the distorsion.

Parameters

- **mode_source** – path to the mode file (by default a pickle file)
- **poscar_gs** – path to the file containing the ground state atomic positions.
- **poscar_es** – path to the file containing the excited state atomic positions.

- **load_phonons** – a function to read mode_source.
- **use_cm1** – use cm1 as the unit for frequency instead of meV.
- **disp** – standard deviation of the gaussians in background in meV.
- **mpl_params** – dictionary of kw parameters for pyplot.plot.
- **mask** – a mask used in other computations to show on the plot.

Returns
(figure, (ax_FC, ax_S))

hylight.multi_phonons.**rect**(*n*)

A dummy windowing function that works like numpy.hamming, but as no effect on data.

hylight.multi_phonons.**sigma_full_nd**(*T*, *delta_R*, *modes_gs*, *modes_es*, *mask=None*)

Compute the width of the ZPL for the ExPES.FULL_ND mode.

Parameters

- **T** – temperature in K.
- **delta_R** – distorsion in A.
- **modes_gs** – list of Modes of the ground state.
- **modes_es** – list of Modes of the excited state.

Returns
np.array with only the width for the modes that are real in ground state.

hylight.multi_phonons.**sigma_hybrid**(*T*, *S*, *e_phonon*, *e_phonon_e*)

Compute the width of the ZPL for the ExPES.SINGLE_ES_FREQ mode.

hylight.multi_phonons.**spectrum**(*mode_source*, *poscar_gs*, *poscar_es*, *zpl*, *T*, *fc_shift_gs=None*, *fc_shift_es=None*, *e_max=None*, *resolution_e=0.0001*, *bias=0*, *mask=None*, *load_phonons=<function load_phonons>*, *pre_convolve=None*, *shape=LineShape.GAUSSIAN*, *omega_eff_type=OmegaEff.FC_MEAN*, *result_store=None*, *ex_pes=<hylight.multi_phonons.ExPES object>*, *correct_zpe=False*)

Compute a complete spectrum without free parameters.

Parameters

- **mode_source** – path to the vibration computation output file (by default a pickled file)
- **path_struct_gs** – path to the ground state relaxed structure file (by default a POSCAR)
- **path_struct_es** – path to the excited state relaxed structure file (by default a POSCAR)
- **zpl** – zero phonon line energy in eV
- **T** – temperature in K
- **fc_shift_gs** – Ground state/absorption Franck-Condon shift in eV
- **fc_shift_es** – Exited state/emmission Franck-Condon shift in eV
- **e_max** – max energy in eV (should be at least > 2*zpl)
- **resolution_e** – energy resolution in eV
- **bias** – ignore low energy vibrations under bias in eV

- **pre_convolve** – (float, optional, None) if not None, standard deviation of the pre convolution gaussian
- **load_phonons** – a function that takes mode_source and produce a list of phonons. By default expect an mode_source.

Returns
(energy_array, intensity_array)

3.2.8 hylight.npz module

Serialization of modes to numpy zipped file.

hylight.npz.**archive_modes**(*modes*, *dest*, *compress=False*)

Store modes in dest using numpy's npz format.

Parameters

- **modes** – a list of Mode objects.
- **dest** – the path to write the modes to.

Returns
the data returned by load_phonons.

hylight.npz.**load_phonons**(*source*)

Load modes from a Hylight archive.

hylight.npz.**load_phonons_npz**(*source*)

hylight.npz.**load_phonons_pickle**(*source*, *gz=False*)

Load modes from a pickled file.

hylight.npz.**pops_and_masses**(*modes*)

3.2.9 hylight.struct module

A generic representation of a crystal cell.

class hylight.struct.**Struct**(*cell_parameters*, *species*, *species_names=None*)

Bases: object

A general description of a periodic crystal cell.

Store all the required infos to describe a given set of atomic positions.

property atoms

List the species names in an order matching *self.raw*.

copy()

property raw

Return an array of atomic positions.

This can be modified overwritten, but not modified in place.

property system_name

The name of the system, eventually generated from formula.

Can be overwritten.

3.2.10 hylight.utils module

Pervasive utilities.

exception hylight.utils.**InputError**

Bases: ValueError

An exception raised when the input files are not as expected.

hylight.utils.**gaussian**(*e*, *sigma*, *standard=True*)

A Gaussian function.

Parameters

- **e** – mean
- **sigma** – standard deviation
- **standard** – if True the curve is normalized to have an area of 1 if False the curve is normalized to have a maximum of 1

hylight.utils.**gen_translat**(*lattice: ndarray*)

Generate all translations to adjacent cells

Parameters
lattice – np.ndarray([a, b, c]) first lattice parameter

hylight.utils.**make_cell**(*val*)

An helper function to create a mutable cell/box.

Parameters
val – initial value

Returns
a function c

Example:

```
>>> c = make_cell(42)
>>> c()
42
>>> c(23)
>>> c()
23
```

hylight.utils.**measure_fwhm**(*x*, *y*)

Measure the full width at half maximum of a given spectrum.

Warning: It may fail if there are more than one band that reach half maximum in the array. In this case you may want to use select_interval to make a window around a single band.

Parameters

- **x** – the energy array
- **y** – the intensity array

Returns
FWHM in the same unit as x.

hylight.utils.**periodic_diff**(*lattice*, *ref*, *disp*)

Compute the displacement between ref and disp, accounting for periodic conditions.

hylight.utils.**select_interval**(*x*, *y*, *emin*, *emax*, *normalize=False*, *npoints=None*)

Extract an interval of a spectrum and return the windows x and y arrays.

Parameters

- **x** – x array
- **y** – y array
- **emin** – lower bound for the window
- **emax** – higher bound for the window
- **normalize** – if true, the result y array is normalized
- **npoints** – if an integer, the result arrays will be interpolated to contains exactly npoints linearly distributed between emin and emax.

Returns

(windowed_x, windowed_y)

3.2.11 Module contents

The Hylight package.

License

hylight.**setup_logging**()

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Hylight] (Ref comming soon)

[BaZrO3:Ti] Cavignac, T.; Jobic, S.; C. Latouche, J. Chem. Theory Comput., 2022, 18, 12, 7714-7721

PYTHON MODULE INDEX

h

INDEX

I

L

M

N

O

P

R

S

T

W