PyBEC Documentation

pybec

Nov 25, 2020

CONTENTS:

CHAPTER

ONE

GETTING STARTED

This page details how to get started with PyBEC.

CHAPTER

TWO

API DOCUMENTATION

2.1 PyBEC

PyBEC Python package for extracting and manipulating Born Effective Charges from QuantumEspresso Output

Modules

pybec.analysis	analysis.py Module for manipulating data from QuantumEspresso Output after Parsing
pybec.output	output.py Module for outputting volumetric data to standard formats
pybec.parsers	parsers.py Module that mainly deals with extracting information from QuantumEspresso Output
pybec.plotters	plotters.py Several convenience functions for plotting Born Effective Charges in various ways.
pybec.utils	utils.py Module with some helpful methods for converting between data structures.

2.1.1 pybec.analysis

analysis.py Module for manipulating data from QuantumEspresso Output after Parsing

Contains all functions for calculating Born Effective Charges

Functions

apply_chunks(interpolator, grid_x, grid_y, ...)	
apply_kriging_chunks(krig_interp, grid_x, ...)	
ave_BEC_dict(elements, BECs)	Create a dictionary where the keys are the element symbols and the values are the average Born Effective Charges for that element.
correct_jumps(arr[, x, jump_quantum, ...])	Correct discontinuities in trajectory
correct_jumps_between_arr(arr1, arr2, ...)	Shift the start value of one array to align with the end value of another.
dipole_field(R, p_vec, centroid)	
find_jumps(arr[, thresh])	Detect where the trajectory has anomolous values.

continues on next page

Table 2 – continued from previous page

find_np_atoms(df_matrix, df_np)	
gen_BEC_df(no_efield, clamped_ion, xyz[, ...])	Generate a pandas dataframe containing the Born Effective Charges of the ions in the unit cell
get_BECs(for_0, for_1, e_field, ...)	Calculate the born effective charges for an array of ions.
get_centroid(atoms_dict[, key])	Calculate the centroid of the atoms of a certian element in a unit cell.
get_dipole_field(coords[, dipole_loc, ...])	Returns the electric field from a point dipole.
get_dipole_field_displaced(coords[, ...])	Returns the electric field from a point dipole.
get_dist_to_centroid(atoms_dict, centroid)	Calculate the centroid of the atoms of a certian element in a unit cell.
get_field_along_d(field_dict[, ...])	Calculate the electric field along a specific direction from the FE results.
get_full_pol(directory)	Gets the trajectory of the polarization.
get_spherical_coords_to_centroid(atoms_dict, ...)	Calculate the centroid of the atoms of a certian element in a unit cell.
grid_krig_execute(krig_execute)	
infer_e_field(for_0, for_1, z_exp[, ...])	Calculate the born effective charges for an array of ions.
infer_local_field(for_0, for_1, z_exp[, ...])	Calculate the born effective charges for an array of ions.
interp_3d(coord_arr, val_arr[, resolution, ...])	
krige_grid_search(BEC_df)	
pad_cell(cell_df, lattice[, pad])	
point_in_hull(point, hull[, tolerance, inc])	https://stackoverflow.com/questions/16750618/whats-an-efficient-way-to-find-if-a-point-lies-in-the-convex-hull-of-a-poi 42165596#42165596
points_in_hull(points, hull[, tolerance, inc])	
select_cell_segment(cell_df, lattice, frac_dir)	Select a portion of the unit cell along a given set of directions.
to_Bohr(coords)	Convert a coordinate dictionary from Angstroms to Bohr
volume(lattice)	Calculate the volume of a 3D unit cell.

pybec.analysis.apply_chunks

pybec.analysis.**apply_chunks**(*interpolator*, *grid_x*, *grid_y*, *grid_z*)

pybec.analysis.apply_kriging_chunks

pybec.analysis.**apply_kriging_chunks**(*krig_interp*, *grid_x*, *grid_y*, *grid_z*, *chunksize=None*)

pybec.analysis.ave_BEC_dict

pybec.analysis.**ave_BEC_dict**(*elements*, *BECs*)

Create a dictionary where the keys are the element symbols and the values are the average Born Effective Charges for that element.

pybec.analysis.correct_jumps

pybec.analysis.**correct_jumps**(*arr*, *x=None*, *jump_quantum=None*, *jump_threshold=10.0*)

Correct discontinuities in trajectory

Either corrects using a best guess to align the slope at the discontinuity with the slope on either side of the discontinuity or using a specified jump size. An integer multiple of this jump is added to the trajectory after the discontinuity to match it closely with the trajectory before.

Parameters

- **arr** (*numpy.ndarray*) – Nx0 array of values in the trajectory of some variable.
- **x** (*numpy.ndarray, optional, default=None*) – Nx0 array defining the spacing between values in the trajectory for calculation of slope. If None, a uniform spacing of 1 is assumed.
- **jump_quantum** (*float, optional, default=None*) – This optionally forces the correction added post-discontinuity to be an integer multiple of this jump_quantum size.
- **jump_threshold** (*float, optional, default=10.*) – Sets the sensitivity for detecting jumps. Larger value increases sensitivity and will try to correct smaller jumps.

Returns Nx0 array of original values, with jumps corrected.

Return type np.ndarray

pybec.analysis.correct_jumps_between_arr

pybec.analysis.**correct_jumps_between_arr**(*arr1*, *arr2*, *jump_quantum*)

Shift the start value of one array to align with the end value of another.

Parameters

- **arr1** (*numpy.ndarray*) – Nx0 array of values in the trajectory of some variable.
- **arr2** (*numpy.ndarray*) – Mx0 array of values in the trajectory of some variable. The function will shift all values in this array such that its start aligns with the end of arr1.
- **jump_quantum** (*float*) – The shift added to arr2 will be an integer multiple of this jump_quantum size.

Returns Mx0 array of shifted arr2 values.

Return type np.ndarray

pybec.analysis.dipole_field

pybec.analysis.**dipole_field**(*R*, *p_vec*, *centroid*)

pybec.analysis.find_jumps

pybec.analysis.**find_jumps**(*arr*, *thresh=10*)
Detect where the trajectory has anomolous values.

Parameters **arr** (*numpy.ndarray*) – Nx0 array of values to check for outliers

Returns Array of indices in which input array has a value larger than thresh

Return type np.ndarray

pybec.analysis.find_np_atoms

pybec.analysis.**find_np_atoms**(*df_matrix*, *df_np*)

pybec.analysis.gen_BEC_df

pybec.analysis.**gen_BEC_df**(*no_efield*, *clamped_ion*, *xyz*, *e_field=0.001*, *e_field_direction=[0, 0, 1]*, *add_forces=False*)
Generate a pandas dataframe containing the Born Effective Charges of the ions in the unit cell

Parameters

- **no_efield** (*string*) – File path of QE output file containing the polarization with no applied electric field
- **clamped_ion** (*string*) – File path of QE output file containing the polarization with applied electric field but ions clamped in place
- **xyz** (*string*) – File path of .xyz - formatted coordinates of unit cell
- **e_field** (*float, optional, default=0.001*) – Electric field strength in au.
- **e_field_direction** (*list, optional, default=[0, 0, 1]*) – Vector of the electric field direction. In positive z-direction by default
- **add_forces** (*bool, optional, default=False*) – If True, include the forces on the ions before and after electric field applied to the output dataframe

Returns Dataframe of ionic coordinates and Born effective charges. Columns are ["Element", "X", "Y", "Z", "BEC"], with optional "Force0", "Force1".

Return type pandas.Dataframe

pybec.analysis.get_BECs

pybec.analysis.**get_BECs**(*for_0*, *for_1*, *e_field*, *e_field_direction*)
Calculate the born effective charges for an array of ions.

Parameters

- **for_0** (*dict*) – Ionic forces in zero field in a dictionary where the keys are the element symbols and the values are the numpy force array for all atoms of that element.
- **for_1** (*dict*) – Ionic forces in applied efield but with clamped ions in a dictionary formatted like for_0.
- **e_field** (*float*) – The magnitude of the applied electric field.
- **e_field_direction** (*list*) – The 3D vector direction of the efield. Ex: [0,0,1] is an electric field in the positive z-direction.

pybec.analysis.get_centroid

pybec.analysis.**get_centroid**(*atoms_dict*, *key='all'*)

Calculate the centroid of the atoms of a certian element in a unit cell.

Parameters

- **atoms_dict** (*dict*) – dictionary of atomic coordinates in Angstroms, where the keys are the element symbols and the values are the numpy coordinate array for all atoms of that element.
- **key** (*str*) – the symbol of the element you would like to calculate the centroid for. If 'all' is given, the centroid of the entire unit cell will be computed.

Returns a numpy array of the x,y, and z coordinates (in Angstroms) of the centroid.

Return type numpy.ndarray

pybec.analysis.get_dipole_field

pybec.analysis.**get_dipole_field**(*coords*, *dipole_loc=[0, 0, 0]*, *p_vec=[0, 0, 1]*, *p=1*, *is_angstrom=True*)

Returns the electric field from a point dipole.

Parameters

- **coords** (*dict*) – Dictionary of atomic coordinates, where the keys are the element symbols, and the values are the numpy coordinate array for all atoms of that element.
- **dipole_loc** (*list or numpy.ndarray*) – The 3D coordinates of the dipole location. Ex: dipole_loc=get_centroid(coords, key='Ag')
- **dipole_loc** (*list or numpy.ndarray*) – The 3D coordinates of the dipole location. Ex: dipole_loc=get_centroid(coords, key='Ag')
- **is_angstrom** (*bool, optional, default : True*) – Indicates whether the input atomic coordinates are in Angstroms (if False, Bohr is assumed)

Returns **field** – Electric field at atomic locations in a dictionary with same format as coords.

Return type dict

pybec.analysis.get_dipole_field_displaced

pybec.analysis.**get_dipole_field_displaced**(*coords*, *dipole_loc=[0, 0, 0]*, *p_vec=[0, 0, 1]*, *q=1*, *d=0.1*, *is_angstrom=True*)

Returns the electric field from a point dipole.

Parameters

- **coords** (*dict*) – Dictionary of atomic coordinates, where the keys are the element symbols, and the values are the numpy coordinate array for all atoms of that element.
- **dipole_loc** (*list or numpy.ndarray*) – The 3D coordinates of the dipole location. Ex: dipole_loc=get_centroid(coords, key='Ag')
- **is_angstrom** (*bool, optional, default : True*) – Indicates whether the input atomic coordinates are in Angstroms (if False, Bohr is assumed)

Returns **field** – Electric field at atomic locations in a dictionary with same format as coords.

Return type dict

pybec.analysis.get_dist_to_centroid

pybec.analysis.**get_dist_to_centroid**(*atoms_dict*, *centroid*)

Calculate the centroid of the atoms of a certian element in a unit cell.

Parameters

- **atoms_dict** (*dict*) – Dictionary of atomic coordinates in Angstroms, where the keys are the element symbols and the values are the numpy coordinate array for all atoms of that element.
- **centroid** (*str*) – Numpy array of the x,y, and z coordinates (in Angstroms) of the centroid.

Returns Dictionary of the distances of each ion in unit cell to the specified centroid. The keys are the element symbols and the values are numpy.ndarrays of the distances for each ion.

Return type dict

pybec.analysis.get_field_along_d

pybec.analysis.**get_field_along_d**(*field_dict*, *sub_mean_field=False*, *e_field=0.25*, *e_field_direction=[0, 0, 1]*)

Calculate the electric field along a specific direction from the FE results.

Parameters

- **field_dict** (*dict*) – Electric field at atomic locations in a dictionary where the keys are the element symbols and the values are the numpy array of the electric field for all atoms of that element.
- **sub_mean_field** (*bool, optional*) – If set, the external applied field is subtracted from the calculated fields, meaning that only the local field disturbance caused by the inclusion will be plotted. Defaults to False.
- **e_field** (*float*) – The magnitude of the applied electric field in V/m.
- **e_field_direction** (*list*) – The 3D vector direction of the efield. Ex: [0,0,1] is an electric field in the positive z-direction.

Returns **field** – Electric field magnitude along the specified direction at atomic locations in a dictionary with same format as field_dict.

Return type dict

pybec.analysis.get_full_pol

pybec.analysis.**get_full_pol**(*directory*)

Gets the trajectory of the polarization.

Parameters **directory** (*str*) – The path to a directory containing all of the output files to parse for polarization data.

Returns Nx2 array with the first column containing the time and second column containing the overall polarization. Duplicate timesteps are removed during processing.

Return type numpy.ndarray

pybec.analysis.get_spherical_coords_to_centroid

pybec.analysis.**get_spherical_coords_to_centroid**(*atoms_dict*, *centroid*)
Calculate the centroid of the atoms of a certian element in a unit cell.

Parameters

- **atoms_dict** (*dict*) – Dictionary of atomic coordinates in Angstroms, where the keys are the element symbols and the values are the numpy coordinate array for all atoms of that element.
- **centroid** (*str*) – Numpy array of the x,y, and z coordinates (in Angstroms) of the centroid.

Returns

- **r** (*dict*) – Dictionary of the distances of each ion in unit cell to the specified centroid. The keys are the element symbols and the values are numpy.ndarrays of the distances for each ion.
- **phi** (*dict*) – Dictionary of the polar angle of each ion in unit cell with reference to an origin at the specified centroid.
- **theta** (*dict*) – Dictionary of the azimuthal angle of each ion in unit cell with reference to an origin at the specified centroid.

pybec.analysis.grid_krig_execute

pybec.analysis.**grid_krig_execute**(*krig_execute*)

pybec.analysis.infer_e_field

pybec.analysis.**infer_e_field**(*for_0*, *for_1*, *z_exp*, *e_field_direction=[0, 0, 1]*)
Calculate the born effective charges for an array of ions.

Parameters

- **for_0** (*dict*) – Ionic forces in zero field in a dictionary where the keys are the element symbols and the values are the numpy force array for all atoms of that element.
- **for_1** (*dict*) – Ionic forces in applied efield but with clamped ions in a dictionary formatted like for_0.
- **z_exp** (*dict*) – Expected born effective charge for each element type from a matrix-only calculation. Keys are element symbols, and values are expected BECs.
- **e_field_direction** (*list, optional, default: [0,0,1]*) – The 3D vector direction of the efield. Ex: [0,0,1] is an electric field in the positive z-direction.

pybec.analysis.infer_local_field

pybec.analysis.**infer_local_field**(*for_0*, *for_1*, *z_exp*, *e_ext=0.001*, *e_field_direction=[0, 0, 1]*)
Calculate the born effective charges for an array of ions.

Parameters

- **for_0** (*dict*) – Ionic forces in zero field in a dictionary where the keys are the element symbols and the values are the numpy force array for all atoms of that element.
- **for_1** (*dict*) – Ionic forces in applied efield but with clamped ions in a dictionary formatted like for_0.

- **z_exp** (*dict*) – Expected born effective charge for each element type from a matrix-only calculation. Keys are element symbols, and values are expected BECs.
- **e_ext** (*float, optional, default: 0.001*) – The magnitude of the applied electric field (au).
- **e_field_direction** (*list, optional, default: [0,0,1]*) – The 3D vector direction of the efield. Ex: [0,0,1] is an electric field in the positive z-direction.

pybec.analysis.interp_3d

pybec.analysis.**interp_3d**(*coord_arr*, *val_arr*, *resolution=100j*, *lattice=None*, *xlim=(0, 1)*, *ylim=(0, 1)*, *zlim=(0, 1)*, *method='linear'*, *return_grids=False*)

pybec.analysis.krige_grid_search

pybec.analysis.**krige_grid_search**(*BEC_df*)

pybec.analysis.pad_cell

pybec.analysis.**pad_cell**(*cell_df*, *lattice*, *pad=0.4*)

pybec.analysis.point_in_hull

pybec.analysis.**point_in_hull**(*point*, *hull*, *tolerance=1e-12*, *inc=False*)
https://stackoverflow.com/questions/16750618/whats-an-efficient-way-to-find-if-a-point-lies-in-the-convex-hull-of-a-point-cl/42165596#42165596

pybec.analysis.points_in_hull

pybec.analysis.**points_in_hull**(*points*, *hull*, *tolerance=1e-12*, *inc=False*)

pybec.analysis.select_cell_segment

pybec.analysis.**select_cell_segment**(*cell_df*, *lattice*, *frac_dir*)
Select a portion of the unit cell along a given set of directions.

The fractional direction is a vector of coefficients for the lattice vectors of the unit cell. For lattice vectors a, b, c, a direction vector of [0.5, 1, 0.5] means that we want the cell up to 0.5a and 0.5c, with all values of b.

A negative value for direction means that we want values greater than (1 - fraction) times the lattice vector. Thus, [-0.3, 1, 1] means that we want the cell segment greater than 0.7a. [-0.3,-0.3,-0.3] would mean we want all cell positions from that lie within [0.7a, a], [0.7b, b], and [0.7c, c].

pybec.analysis.to_Bohr

pybec.analysis.**to_Bohr**(*coords*)
Convert a coordinate dictionary from Angstroms to Bohr

pybec.analysis.volume

pybec.analysis.**volume**(*lattice*)
Calculate the volume of a 3D unit cell.

Parameters **lattice** (*numpy.ndarray*) – 3 x 3 array with the unit cell vectors as the rows. That is numpy.array([[a1, a2, a3], [b1, b2, b3], [c1, c2, c3]])

Returns Volume of unit cell, calculated as (a x b) * c

Return type float

2.1.2 pybec.output

output.py Module for outputting volumetric data to standard formats

Includes functions to output 3D grid data to the XCrysden file format, which can be read in by XCrysden, VESTA, or other visualization software.

Functions

add_datagrid_3d(input_filename, ... [, name])	
arr_2d_string(arr)	
col_major_string(v)	
indent(string, num_spaces)	
print_coordinates(coords, xyz_file[, ...])	Prints the XYZ coordinates to a specified file.

pybec.output.add_datagrid_3d

pybec.output.**add_datagrid_3d**(*input_filename*, *output_filename*, *data*, *span_vectors*, *name='my3d_data'*)

pybec.output.arr_2d_string

pybec.output.**arr_2d_string**(*arr*)

pybec.output.col_major_string

pybec.output.**col_major_string**(*v*)

pybec.output.indent

pybec.output.**indent**(*string*, *num_spaces*)

pybec.output.print_coordinates

pybec.output.**print_coordinates**(*coords*, *xyz_file*, *comment=''*, *format='block'*, *unit='angstrom'*)

Prints the XYZ coordinates to a specified file.

Parameters

- **coords** (*collections.OrderedDict*) – The coordinates in angstroms in a dictionary where the keys are the element symbols and the values are the numpy arrays of the coordinates for all atoms of that element.
- **xyz_file** (*str*) – File path to the output .xyz file
- **comment** (*str*) – Comment to include in line 2 of xyz file

Returns

Return type None

2.1.3 pybec.parsers

parsers.py Module that mainly deals with extracting information from QuantumEspresso Output

Handles the primary functions

Functions

get_converged_forces(output_file)	Retrieves the converged forces from the last step of the specified output file.
get_coordinates(xyz_file[, skip, delimiter])	Loads the XYZ coordinates of the specified file.
get_dipole(output_file)	Retrieves the electric and ionic polarizations from output file.
get_final_cell(output_file[, unit_multiplier])	Retrieves the final cell vectors from the last step of the specified output file.
get_final_positions(output_file[, ...])	Retrieves the final positions from the last step of the specified output file.
get_full_dipole(directory[, remove_dupes])	Gets the trajectory of the polarization.
get_lattice(xyz_file)	Loads the lattice vectors from the specified file.
get_trajectory(output_file[, unit_multiplier])	Retrieves the final positions from the last step of the specified output file.

pybec.parsers.get_converged_forces

pybec.parsers.**get_converged_forces**(*output_file*)

Retrieves the converged forces from the last step of the specified output file.

Parameters **output_file** (*str*) – File path to the output file for the step from which to pull converged forces.

Returns Ionic forces in a dictionary where the keys are the element symbols and the values are the numpy force array for all atoms of that element.

Return type dict

pybec.parsers.get_coordinates

pybec.parsers.**get_coordinates**(*xyz_file*, *skip=2*, *delimiter=None*)

Loads the XYZ coordinates of the specified file.

Parameters

- **xyz_file** (*str*) – File path to the .xyz file containing the relaxed crystal structure
- **delimiter** (*str*) – delimiter separating x, y, z coordinates in the file, to be used by numpy's genfromtxt method

Returns The coordinates in angstroms in a dictionary where the keys are the element symbols and the values are the numpy arrays of the coordinates for all atoms of that element.

Return type dict

pybec.parsers.get_dipole

pybec.parsers.**get_dipole**(*output_file*)

Retrieves the electric and ionic polarizations from output file.

Parameters **output_file** (*str*) – File path to the output file for the step from which to pull polarizations.

Returns header, pols. Header is a list of the header names associated with pols. pols is an N X 5 np.ndarray. For each of N steps, it contains The associated step number, timestep, electronic, ionic, and total cell dipole.

Return type tuple

pybec.parsers.get_final_cell

pybec.parsers.**get_final_cell**(*output_file*, *unit_multiplier=1.0*)

Retrieves the final cell vectors from the last step of the specified output file.

Parameters **output_file** (*str*) – File path to the output file for the step from which to pull converged forces.

Returns A 3x3 array of the cell vectors, as 3 row vectors [[a1,a2,a3],[b1,b2,b3],[c1,c2,c3]]

Return type np.ndarray

pybec.parsers.get_final_positions

pybec.parsers.**get_final_positions**(*output_file*, *unit_multiplier=1.0*)
Retrieves the final positions from the last step of the specified output file.

Parameters **output_file** (*str*) – File path to the output file for the step from which to pull converged forces.

Returns Ionic coordinates in a dictionary where the keys are the element symbols and the values are the numpy coordinate array for all atoms of that element.

Return type dict

pybec.parsers.get_full_dipole

pybec.parsers.**get_full_dipole**(*directory*, *remove_dupes=True*)
Gets the trajectory of the polarization.

Parameters

- **directory** (*str*) – The path to a directory containing all of the output files to parse for polarization data.
- **remove_dupes** (*bool*) – If true, keep only one of each step number, the one from the last file it occurs in.

Returns Nx2 array with the first column containing the time and second column containing the overall polarization. Duplicate timesteps are removed during processing.

Return type numpy.ndarray

pybec.parsers.get_lattice

pybec.parsers.**get_lattice**(*xyz_file*)
Loads the lattice vectors from the specified file.

Parameters **xyz_file** (*str*) – File path to the .xyz file containing the relaxed crystal structure

Returns The lattice vectors in angstroms in a numpy array where the rows are lattice vectors a, b, and c, and the columns are the unit directions x, y, z.

Return type numpy.ndarray

pybec.parsers.get_trajectory

pybec.parsers.**get_trajectory**(*output_file*, *unit_multiplier=1.0*)
Retrieves the final positions from the last step of the specified output file.

Parameters **output_file** (*str*) – File path to the output file for the step from which to pull trajectory.

Returns Ionic coordinates at each timestep in a dictionary where the keys are the element symbols and the values are the numpy coordinate array for all atoms of that element.

Return type dict

2.1.4 pybec.plotters

plotters.py Several convenience functions for plotting Born Effective Charges in various ways.

Functions

add_colorbar(fig, plot, cbar_pos[, shrink])	
multi_slice_viewer(volume, fig)	
multi_slice_viewer_BEC(fig, volume, lattice)	
next_direction(ax)	Go to the next slice direction.
next_slice(ax)	Go to the next slice.
plot_BEC_3D(no_efield, clamped_ion, xyz, ...)	
plot_BEC_heatmap_slices(fig, no_efield, ...)	
plot_BEC_v_spher(no_efield, clamped_ion, ...)	Plots the Born Effective Charges for each ion against the distance of that ion from the nanoparticle centroid.
plot_FE_E_v_spher(xyz, xyzE, matrix_name, ...)	Plots the electric field predicted on each ion by a continuum (FE) approach.
plot_atoms(ax)	
previous_slice(ax)	Go to the previous slice.
process_key(event)	
remove_keymap_conflicts(new_keys_set)	

pybec.plotters.add_colorbar

pybec.plotters.**add_colorbar**(*fig*, *plot*, *cbar_pos*, *shrink=0.5*)

pybec.plotters.multi_slice_viewer

pybec.plotters.**multi_slice_viewer**(*volume*, *fig*)

pybec.plotters.multi_slice_viewer_BEC

pybec.plotters.**multi_slice_viewer_BEC**(*fig*, *volume*, *lattice*, *add_atoms=True*, *cell=None*, *color_dict=None*, *marker_dict=None*, *cmap_atoms=False*, *cmap='viridis'*, *ion_cmap='plasma'*, *num_bins=6*, *resolution=50*, *is_kriging=False*)

pybec.plotters.next_direction

pybec.plotters.**next_direction**(*ax*)

Go to the next slice direction.

pybec.plotters.next_slice

pybec.plotters.**next_slice**(*ax*)
Go to the next slice.

pybec.plotters.plot_BEC_3D

pybec.plotters.**plot_BEC_3D**(*no_efield*, *clamped_ion*, *xyz*, *matrix_name*, *np_name*, *e_field=0.001*, *e_field_direction=2*, *cbar_pos='top'*, *legend=True*, *figsize=(8, 8)*, *marker_dict=None*, *grid=False*, *cmap='magma'*, *alpha=1.0*, *cbar_shrink=1.0*)

pybec.plotters.plot_BEC_heatmap_slices

pybec.plotters.**plot_BEC_heatmap_slices**(*fig*, *no_efield*, *clamped_ion*, *xyz*, *matrix_name*, *np_name*, *matrix_no_efield*, *matrix_clamped_ion*, *matrix_xyz*, *e_field=0.001*, *e_field_direction=2*, *interpolation='linear'*, *cbar_pos='top'*, *legend=True*, *marker_dict=None*, *pad=0.2*, *color_dict=None*, *grid=False*, *cmap='magma'*, *ion_cmap='plasma'*, *num_bins=6*, *track_slices=True*, *res=100*, *add_atoms=True*, *cmap_atoms=False*, *cbar_shrink=1.0*)

pybec.plotters.plot_BEC_v_spher

pybec.plotters.**plot_BEC_v_spher**(*no_efield*, *clamped_ion*, *xyz*, *matrix_name*, *np_name*, *np_element*, *to_plot='r'*, *centroid_of='all'*, *e_field=0.001*, *e_field_direction=[0, 0, 1]*, *cmap=None*, *cbar_pos='top'*, *legend=True*, *figsize=(8, 4)*, *marker_dict=None*, *color_dict=None*, *grid=False*, *alpha=1.0*, *cbar_shrink=1.0*)

Plots the Born Effective Charges for each ion against the distance of that ion from the nanoparticle centroid.

Parameters

- **no_efield** (*str*) – File path to the zero field step output file
- **clamped_ion** (*str*) – File path to the clamped ion step output file
- **xyz** (*str*) – File path to the optimized structure .xyz file
- **matrix_name** (*str*) – Whatever you want to call the matrix, e.g. 'MgO'
- **np_name** (*str*) – Whatever you want to call the mNP, e.g. 'Ag8_333'
- **np_element** (*str*) – Elemental symbol for nanoparticle element.
- **centroid_of** (*str, optional*) – The element you want to calculate the centroid for when calculating and plotting against the distance to centroid. If 'all', it will calculate the centroid of the whole unit cell. 'Ag' will calculate the centroid of all silver ions in the unit cell in the relaxed structure. Default: 'all'
- **e_field** (*float, optional*) – The electric field in au. Default: 0.001
- **e_field_direction** (*int, optional*) – The direction the field is applied, where the x-axis is [1,0,0], y-axis is [0,1,0], and z-axis is [0,0,1]. Default: [0,0,1]

- **cmap** (*string, optional*) – The matplotlib colormap style used to color the absolute value of their Born Effective Charge. If None, the data instead will be colored by element using the color_dict colors or default matplotlib ones. Default: None
- **cbar_pos** (*str, optional*) – Where to place the colorbar if cmap is used. Can be 'top', 'right', or 'bottom'. Default: 'top'
- **legend** (*bool, optional*) – Whether to include a legend labelling the different elements. Default: True
- **figsize** ((*int, int), optional*) – Width, Height of the figure. Default: (8,4)
- **marker_dict** (*dict, optional*) – Custom dictionary specifying which markers to use for each element, where the keys are element symbols and the values are valid matplotlib marker symbols. Default: None Ex: {'O': 'o', 'Ag': 'x', 'Mg': '>'}
- **color_dict** (*dict, optional*) – Custom dictionary specifying which colors to use for each element, where the keys are element symbols and the values are valid matplotlib color symbols. Default: None Ex: {'O': 'r', 'Ag': 'k', 'Mg': 'y'}
- **grid** (*bool, optional*) – Whether to include grid lines in the plot. Default: False
- **alpha** (*float*) – The alpha channel value for the plot colors. Default: 1.0

Returns

Return type None

pybec.plotters.plot_FE_E_v_spher

pybec.plotters.**plot_FE_E_v_spher**(*xyz*, *xyzE*, *matrix_name*, *np_name*, *np_element*, *to_plot='r'*, *centroid_of='all'*, *sub_mean_field=False*, *e_field=0.25*, *e_field_direction=[0, 0, 1]*, *center_dict=None*, *ref_neg=True*, *cmap=None*, *cbar_pos='top'*, *legend=True*, *figsize=(8, 4)*, *marker_dict=None*, *color_dict=None*, *grid=False*, *alpha=1.0*, *cbar_shrink=1.0*, *units='au'*)

Plots the electric field predicted on each ion by a continuum (FE) approach.

Parameters

- **xyz** (*str*) – File path to the optimized structure .xyz file
- **xyzE** (*str or dict*) – File path to the file containing the FE efield for each atomic coordinate, or Dictionary of electric field values, where the keys are the element symbols, and the values are the Nx3 numpy array of electric field values for all atoms of that element. Ex: xyzE=get_dipole_field(get_coordinates(xyz), get_centroid(blah, key='Ag'), q=1)
- **matrix_name** (*str*) – Whatever you want to call the matrix, e.g. 'MgO'
- **np_name** (*str*) – Whatever you want to call the mNP, e.g. 'Ag8_333'
- **centroid_of** (*str, optional*) – The element you want to calculate the centroid for when calculating and plotting against the distance to centroid. If 'all', it will calculate the centroid of the whole unit cell. 'Ag' will calculate the centroid of all silver ions in the unit cell in the relaxed structure. defaults to 'all'
- **sub_mean_field** (*bool, optional*) – If set, the external applied field is subtracted from the calculated fields, meaning that only the local field disturbance caused by the inclusion will be plotted. defaults to True
- **e_field** (*float, optional*) – The applied electric field in V/m. Default: 0.25

- **e_field_direction** (*int, optional*) – The direction the field is applied, where the x-axis is [1,0,0], y-axis is [0,1,0], and z-axis is [0,0,1]. Default: [0,0,1]
- **center_dict** (*dict, optional*) – Custom dictionary specifying where to center each element on the y-axis of the plot, where the keys are element symbols and the values are floats or integers. Default: None Ex: {'O': -2, 'Mg': 2, 'Ag': 0}
- **cmap** (*string, optional*) – The matplotlib colormap style used to color the absolute value of their Born Effective Charge. If None, the data instead will be colored by element using the color_dict colors or default matplotlib ones. Default: None
- **cbar_pos** (*str, optional*) – Where to place the colorbar if cmap is used. Can be 'top', 'right', or 'bottom'. Default: 'top'
- **legend** (*bool, optional*) – Whether to include a legend labelling the different elements. Default: True
- **figsize** (*(int, int), optional*) – Width, Height of the figure. Default: (8,4)
- **marker_dict** (*dict, optional*) – Custom dictionary specifying which markers to use for each element, where the keys are element symbols and the values are valid matplotlib marker symbols. Default: None Ex: {'O': 'o', 'Ag': 'x', 'Mg': '>'}
- **color_dict** (*dict, optional*) – Custom dictionary specifying which colors to use for each element, where the keys are element symbols and the values are valid matplotlib color symbols. Default: None Ex: {'O': 'r', 'Ag': 'k', 'Mg': 'y'}
- **grid** (*bool, optional*) – Whether to include grid lines in the plot. Default: False
- **alpha** (*float, optional*) – The alpha channel value for the plot colors. Default: 1.0
- **units** (*str, optional*) – The units for electric field to be added to the y-axis. Default: "au"

Returns

Return type None

pybec.plotters.plot_atoms

pybec.plotters.**plot_atoms**(*ax*)

pybec.plotters.previous_slice

pybec.plotters.**previous_slice**(*ax*)

Go to the previous slice.

pybec.plotters.process_key

pybec.plotters.**process_key**(*event*)

pybec.plotters.remove_keymap_conflicts

pybec.plotters.**remove_keymap_conflicts**(*new_keys_set*)

2.1.5 pybec.utils

utils.py Module with some helpful methods for converting between data structures.

The rest of the package often utilizes atomic data in the form of either pandas Dataframes or dictionaries with element symbols for keys. Functions in this module convert between these two representations

Functions

as_dataframe(atoms_dict[, BEC, for0, for1])	make a pandas dataframe with the combined coordinates
df_to_dicts(df)	
make_sphere(r, centr, N)	Adapted from Brian Z Bentz (2020).

pybec.utils.as_dataframe

pybec.utils.**as_dataframe**(*atoms_dict*, *BEC=None*, *for0=None*, *for1=None*)

make a pandas dataframe with the combined coordinates

pybec.utils.df_to_dicts

pybec.utils.**df_to_dicts**(*df*)

pybec.utils.make_sphere

pybec.utils.**make_sphere**(*r*, *centr*, *N*)

Adapted from Brian Z Bentz (2020). mySphere(N) (https://www.mathworks.com/matlabcentral/fileexchange/57877-mysphere-n)

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

R

S

T

V