surfaxe

Release v0.1

Katarina Brlec, Daniel Davies

Oct 01, 2021

CONTENTS

View the code on Github here.

Contents:

CHAPTER ONE

GETTING STARTED

1.1 Introduction

The purpose of *surfaxe* is to simplify the calculation of surface properties of inorganic compounds using first-principles codes. The package includes pre-processing tools to cleave surfaces from the bulk, automatically set up calculation directories and facilitate convergence testing, as well as post-processing tools to calcualte surface energies, electrostatic potentials, and produce publication-qaulity plots.

1.2 Installation

Installation will be via `pip` in the near future. For now, please clone the repository and install the latest stable version:

```
git clone https://github.com/SMTG-UCL/surfaxe.git
cd surfaxe
pip install -e .
```

The `-e` is optional and will install the project in developer (editable) mode.

For the code to generate VASP input files along with the surface slabs, pymatgen POTCAR environment must be set up correctly.

1.3 Usage

In general, there are two ways to use *surfaxe*: (i) at the command line or (ii) using python in scripts or notebooks.

Take a look at our command line interface (CLI) examples for an overview of the CLI tools available.

There is full documentation for all modules (have a browse of the side bar on the left) if you would rather use the python API directly.

CHAPTER

TWO

COMMAND LINE INTERFACE (CLI)

While *surfaxe* has a full python API (see our tutorials page for example usage) it also has an intuitive command line interface (CLI). Below are some simple examples of what you can do with *surfaxe* at the command line.

You can get a full list of accepted flags and what they do for each command using the `--help` or `-h` flag, e.g.:

```
$ surfaxe-bonds -h

> usage: surfaxe-bonds [-h] [-s STRUCTURE] [-b BOND [BOND ...]]
                 [--oxi-list OX_STATES_LIST [OX_STATES_LIST ...]]
                 [--oxi-dict OX_STATES_DICT] [--no-csv]
                 [--csv-fname CSV_FNAME] [--no-plot]
                 [--plt-fname PLT_FNAME] [-c COLOR] [--width WIDTH]
                 [--height HEIGHT] [--dpi DPI] [--yaml]

Parses the structure looking for bonds between atoms. Check the validity of
the nearest neighbour method on the bulk structure before using it on slabs.

optional arguments:
-h, --help            show this help message and exit
-s STRUCTURE, --structure STRUCTURE
                      Filename of structure file in any format supported by
                      pymatgen (default: POSCAR
-b BOND [BOND ...], --bond BOND [BOND ...]
                      List of elements e.g. Ti O for a Ti-O bond
--oxi-list OX_STATES_LIST [OX_STATES_LIST ...]
                      Add oxidation states to the structure as a list
                      e.g. 3 3 -2 -2 -2
--oxi-dict OX_STATES_DICT
                      Add oxidation states to the structure as a dictionary
                      e.g. Fe:3,O:-2
--no-csv              Prints data to terminal
--csv-fname CSV_FNAME
                      Filename of the csv file (default: bond_analysis.csv)
--no-plot             Turns off plotting
--plt-fname PLT_FNAME
                      Filename of the plot (default: bond_analysis.png)
-c COLOR, --color COLOR
                      Color of the marker in any format supported by mpl
                      e.g. "#eeefff" hex colours starting with # need to be
                      surrounded with quotation marks
--width WIDTH         Width of the figure in inches (default: 6)
```

(continues on next page)

(continued from previous page)

```
--height HEIGHT       Height of the figure in inches (default: 5)
--dpi DPI             Dots per inch (default: 300)
--yaml YAML           Read all args from a yaml config file. Completely
                      overrides any other flags set
```

The behaviour of default parameters of the functions is extensively documented in the `surfaxe` python package section of the docs.

2.1 Pre-processing commands

surfaxe-generate-slabs: Generates all unique slabs with specific Miller indices or up to a maximum Miller index for a set of slab and vacuum thicknesses.

Example: `surfaxe-generate -s bulk_structure.cif --hkl 1,1,0 -t 20 40 -v 20 40 -f` generates all slabs for the (1,1,0) direction for minimum slab and vacuum thicknesses of 20 Å and 40 Å. The `-f` option organises these into subdirectories with all required VASP input files required to run singleshot calculations uisng default settings. It includes all combinations for zero-dipole terminations with inversion symmetry. The directory structure produced is:

```
100/              <-- Miller index
  ├── 20_20_0/    <-- slab-thickness_vacuum-thickness_termination-number
  ├── 20_40_0/
  ├── 40_20_0/
  └── 40_40_0/
      ├── POSCAR  <-- VASP files
      ├── INCAR
      ├── POTCAR
      └── KPOINTS
```

Note: The hkl flag must be comma-separated with no spaces and the list of thicknesses and vacuums must be space-separated.

Note: To use the :mod:`-f` option you must first set up the `pymatgen POTCAR environment <https://pymatgen.org/installation.html#potcar-setup>`_.

Similarly, to above the script can be modified to consider multiple Miller indices.

Example: `surfaxe-generate -s bulk_structure.cif --hkl 1,1,0 1,1,1 -t 20 40 -v 20 40 -f` generates all (1,1,0) and (1,1,1) slabs with minimum slab and vacuum thicknesses of 20 Å and 40 Å.

Note: h,k,l are comma-separated with no spaces, while the two (or more) Miller indices are space-separated.

Lastly, a maximum hkl value can be supplied as an integer so that the script finds all zero-dipole slabs up to that maximum Miller index.

Example: `surfaxe-generate -s SnO2.cif --hkl 2 -t 20 40 -v 30` generates all slabs with Miller indices up to a maximum value of 2, with minimum slab thicknesses of 20 Å and of 40 Å, and minimum vacuum thickness of 30 Å.

2.2 Post-processing commands

surfaxe-parse-energies: Parses data produced by electronic structure codes once calculations have been run in then directory structures produced by the pre-processing commands. Can optionally collect vacuum and core energies.

Example: `surfaxe-parse-energies --hkl 0,0,1 -b 8.83099` saves a csv file of surface energies and energies per atom for each slab-vacuum combination. See the Tutorials directory for examples.

surfaxe-plot-surfen and **surfaxe-plot-enatom** can be used to customise the surface energy and energy per atom plots independetnly based on the data already collated with **surfaxe-parse-energies**.

surfaxe-parse-structures: Parses the (relaxed) structures from convergence calculations and collates them into the same json format as is created when surface slabs are generated. Can optionally perform bond analysis for multiple specified bonds. Useful for comparison of relaxed and unrelaxed surfaces slabs and determination of convergence.

2.3 Analysis commands

surfaxe-potential: Reads the local electrostatic potential file and plots the planar and macroscopic averages normal to the surface. Currently only the VASP LOCPOT file is supported as input.

Example: `surfaxe-potential -l LOCPOT -v 11.5` produces a plot assuming a lattice vector of 11.5 Angstroms and saves the plot data to a csv file.

surfaxe-bonds: Analyse bonding in the structure using Pymatgen's local_env module. Average bond lengths for each pair of species of interest can be plotted as a function of c lattice vector (normal to the slab surface). This can be useful for checking whether the center of the slab has converged, where bond distances should be bulk-like.

Example: `surfaxe-bonds -s CONTCAR -b Sn O` plots the average Sn-O bond length from the VASP output structure file. A csv file of the data plotted is also produced.

surfaxe-plot-potential and **surfaxe-plot-bonds** can be used to generate the plots based on the data collated with **surfaxe-potential** and **surfaxe-bonds**, allowing customisation of plots without having to re-analyse the data. All plotting functionality is accessible through the main functions as well.

surfaxe-simplenn and **surfaxe-complexnn**: Analyse the bonding in the slab, again using Pymatgen functions. *simplenn* is faster, but less reliable for systems with more complex bonding. *complexnn* is more robust but requires a dictionary of cutoff bond lengths to be supplied for each pair of species. See the analysis tutorial for further explanation.

Example: `surfaxe-complexnn -s CONTCAR_bivo4 -b Bi3+ O2- 2.46 V5+ O2- 1.73` will analyse the coordination of atoms in this BiVO4 slab and save them to a csv file.

2.4 Data commands

There are some simple convenience commands that can also be used to extract key values from raw data files produced by solid state codes. Currently only commands relating to VASP output files are included, which rely on the surfaxe `vasp_data` module. We hope to expand this in the future.

surfaxe-vacuum and **surfaxe-core** can be used to extract vacuum and core energies, respectively, that are needed to calculate absolute electron energies (ionisation potential and electron affinity). See the Macrodensity tutorials for more information on the steps needed to do this.

2.5 YAML input files

Most CLI commands allow use of YAML input files containing all the arguments which cannot be used in conjunction with other command line argument flags. This is done by specifying the `--yaml` flag which overrides any other flags set in command line by loading the `surfaxe_config.yaml` file.

Sample YAML input files for each of the functions, with defaults and comments are in the `surfaxe/cli/templates` folder. All `**kwargs` of the main function can be passed in the YAML file.

Example: Generation of (1,0,1) CdTe slabs could easily customised so that all VASP input files are created with specific INCAR tags using the following config.yaml file:

```yaml
structure: CdTe.cif
hkl: (1,0,1)
thicknesses: [20, 40]
vacuums: [20, 40]
make_fols: True
make_files: True
max_size: 500
center_slab: True
ox_states:
  Cd: 2
  Te: -2
fmt: poscar
name: POSCAR
config_dict: PBE_config.json
user_incar_settings:
  ENCUT: 460
  KPAR: 3
user_kpoints_settings:
  reciprocal_density: 35
```

The slabs would then be generated using `surfaxe-gethkl --yaml config.yaml`

CHAPTER

THREE

SURFAXE PYTHON PACKAGE

Please peruse the submodules at your leisure

3.1 Submodules

3.1.1 surfaxe.generation module

`surfaxe.generation.`**`generate_slabs`**(*structure*, *hkl*, *thicknesses*, *vacuums*, *save_slabs=True*, *save_metadata=True*, *json_fname=None*, *make_fols=False*, *make_input_files=False*, *max_size=500*, *center_slab=True*, *ox_states=None*, *is_symmetric=True*, *layers_to_relax=None*, *fmt='poscar'*, *name='POSCAR'*, *config_dict=None*, *user_incar_settings=None*, *user_kpoints_settings=None*, *user_potcar_settings=None*, *parallelise=True*, ***kwargs*)

Generates all unique slabs for a specified Miller indices or up to a maximum Miller index with minimum slab and vacuum thicknesses. It includes all combinations for multiple zero dipole symmetric terminations for the same Miller index.

The function returns None by default and generates either:

(i) POSCAR_hkl_slab_vac_index.vasp (default)

(ii) hkl/slab_vac_index folders with structure files

(iii) hkl/slab_vac_index with all VASP input files

Or if *save_slabs=False* a list of dicts of all unique slabs is returned.

Parameters

- **`structure`** (*str* or pmg Structure obj) – Filename of structure file in any format supported by pymatgen or pymatgen structure object.
- **`hkl`** (*tuple*, *list* or *int*) – Miller index as tuple, a list of Miller indices or a maximum index up to which the search should be performed. E.g. if searching for slabs up to (2,2,2) `hkl=2`
- **`thicknesses`** (*list*) – The minimum size of the slab in Angstroms.
- **`vacuums`** (*list*) – The minimum size of the vacuum in Angstroms.
- **`save_slabs`** (*bool*, optional) – Whether to save the slabs to file. Defaults to `True`.
- **`save_metadata`** (*bool*, optional) – Whether to save the slabs' metadata to file. Saves the entire slab object to a json file Defaults to `True`.

- **`json_fname`** (*str*, optional) – Filename of json metadata file. Defaults to bulk_formula_metadata.json
- **`make_fols`** (*bool*, optional) – Makes folders for each termination and slab/vacuum thickness combinations containing structure files.
  - `True`: A Miller index folder is created, in which folders named slab_vac_index are created to which the relevant structure files are saved.

    E.g. for a (0,0,1) slab of index 1 with a slab thickness of 20 Å and vacuum thickness of 30 Å the folder structure would be: `001/20_30_1/POSCAR`
  - `False`: The indexed structure files are put in a folder named after the bulk formula.

    E.g. for a (0,0,1) MgO slab of index 1 with a slab thickness of 20 Å and vacuum thickness of 30 Å the folder structure would be: `MgO/POSCAR_001_20_30_1`

  Defaults to `False`.
- **`make_input_files`** (*bool*, optional) – Makes INCAR, POTCAR and KPOINTS files in each folder. If `make_input_files` is `True` but `make_files` or `save_slabs` is `False`, files will be saved to folders regardless. This only works with VASP input files, other formats are not yet supported. Defaults to `False`.
- **`max_size`** (*int*, optional) – The maximum number of atoms in the slab specified to raise warning about slab size. Even if the warning is raised, it still outputs the slabs regardless. Defaults to `500`.
- **`center_slab`** (*bool*, optional) – The position of the slab in the simulation cell.
  - `True`: the slab is centered with equal amounts of vacuum above and below.
  - `False`: the slab is at the bottom of the simulation cell with all of the vacuum on top of it.

  Defaults to True.
- **`ox_states`** (`None`, *list* or *dict*, optional) – Add oxidation states to the bulk structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`
  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`
  - if `None`: The oxidation states are added by guess.

  Defaults to `None`.
- **`is_symmetric`** (*bool*, optional) – Whether the slabs cleaved should have inversion symmetry. If bulk is non-centrosymmetric, `is_symmetric` needs to be `False` - the function will return no slabs as it looks for inversion symmetry. Take care checking the slabs for mirror plane symmetry before just using them. Defaults to `True`.
- **`layers_to_relax`** (*int*, optional) – Specifies the number of layers at the top and bottom of the slab that should be relaxed, keeps the centre constrained using selective dynamics. NB only works for VASP files
- **`fmt`** (*str*, optional) – The format of the output structure files. Options include 'cif', 'poscar', 'cssr', 'json', not case sensitive. Defaults to 'poscar'.

- **name** (*str*, optional) – The name of the surface slab structure file created. Case sensitive. Defaults to 'POSCAR'
- **config_dict** (*dict* or *str*, optional) – Specifies the dictionary used for the generation of the input files. Suppports already loaded dictionaires, yaml and json files. Surfaxe-supplied dictionaries are PBE (`pe`), PBEsol (`ps`) and HSE06 (`hse06`) for single shot calculations and PBE (`pe_relax`) and PBEsol (`ps_relax`) for relaxations. Not case sensitive. Defaults to PBEsol (`ps`).
- **user_incar_settings** (*dict*, optional) – Overrides the default INCAR parameter settings. Defaults to `None`.
- **user_kpoints_settings** (*dict* or Kpoints object, optional) – Overrides the default kpoints settings. If it is supplied as *dict*, it should be as `{'reciprocal_density': 100}`. Defaults to `None`.
- **user_potcar_settings** (*dict*, optional) – Overrides the default POTCAR settings. Defaults to `None`.
- **parallelise** (*bool*, optional) – Use multiprocessing to generate slabs. Defaults to `True`.

Returns None (default) or unique_slabs (list of dicts)

`surfaxe.generation.`**`oxidation_states`**(*structure*, *ox_states=None*)

Adds oxidation states to the structure object if not already present

Parameters

- **structure** (*obj*) – Pymatgen structure object
- **ox_states** (`None`, *list* or *dict*, optional) – Add oxidation states to the structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`
  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`
  - if `None`: The oxidation states are added by guess.

Returns Structure decorated with oxidation states

3.1.2 surfaxe.convergence module

3.1.3 surfaxe.analysis module

`surfaxe.analysis.`**`bond_analysis`**(*structure*, *bond*, *nn_method=<Mock name='mock()' id='140428077570128'>*, *ox_states=None*, *save_csv=True*, *csv_fname='bond_analysis.csv'*, *save_plt=False*, *plt_fname='bond_analysis.png'*, ***kwargs*)

Parses the structure looking for bonds between atoms. Check the validity of the nearest neighbour method on the bulk structure before using it on slabs.

Parameters

- **structure** (*str*) – filename of structure, takes all pymatgen-supported formats, including pmg structure object

- **bond** (*list*) – Bond to analyse e.g. `['Y', 'O']`
- **nn_method** (*class*, optional) – The coordination number prediction algorithm used. Because the `nn_method` is a class, the class needs to be imported from `pymatgen.analysis.local_env` before it can be instantiated here. Defaults to `CrystalNN()`.
- **ox_states** (`None`, *list* or *dict*, optional) – Add oxidation states to the structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`
  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`
  - if `None`: The oxidation states are added by guess.

  Defaults to `None`.
- **save_csv** (*bool*, optional) – Makes a csv file with the c coordinate of the first atom and bond length. Defaults to `True`.
- **csv_fname** (*str*, optional) – Filename of the csv file. Defaults to `'bond_analysis.csv'`.
- **save_plt** (*bool*, optional) – Make and save the bond analysis plot. Defaults to `False`.
- **plt_fname** (*str*, optional) – Filename of the plot. Defaults to `'bond_analysis.png'`.

Returns DataFrame with the c coordinate of the first atom and bond length

`surfaxe.analysis.`**cart_displacements**(*start*, *end*, *max_disp=0.1*, *save_txt=True*, *txt_fname='cart_displacements.txt'*)

Produces a text file with all the magnitude of displacements of atoms in Cartesian space

Parameters

- **start** (*str*) – Filename of initial structure file in any format supported by pymatgen or pymatgen structure object.
- **end** (*str*) – Filename of final structure file in any format supported by pymatgen or pymatgen structure object.
- **max_disp** (*float*, optional) – The maximum displacement shown. Defaults to 0.1 Å.
- **save_txt** (*bool*, optional) – Save the displacements to file. Defaults to `True`.
- **txt_fname** (*str*, optional) – Filename of the csv file. Defaults to `'cart_displacement.txt'`.

Returns None (default) or DataFrame of displacements of atoms in Cartesian space

`surfaxe.analysis.`**complex_nn**(*start*, *cut_off_dict*, *end=None*, *ox_states=None*, *save_csv=True*, *csv_fname='nn_data.csv'*)

Finds the nearest neighbours for more complex structures. Uses CutOffDictNN() class as the nearest neighbour method. Check validity on bulk structure before applying to surface slabs.

The `site_index` in the produced DataFrame or csv file is one-indexed and represents the atom index in the structure.

Parameters

- **start** (*str*) – filename of structure, takes all pymatgen-supported formats.

- `cut_off_dict` (*dict*) – Dictionary of bond lengths. The bonds should be specified with the oxidation states

  e.g. `{('Bi3+', 'O2-'): 2.46, ('V5+', 'O2-'): 1.73}`

- `end` (*str*, optional) – filename of structure to analyse, use if comparing initial and final structures. The structures must have same constituent atoms and number of sites. Defaults to `None`.
- `ox_states` (`None`, *list* or *dict*, optional) – Add oxidation states to the structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`

  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`

  - if `None`: The oxidation states are added by guess.

  Defaults to `None`

- `save_csv` (*bool*, optional) – Save to a csv file. Defaults to `True`.
- `csv_fname` (*str*, optional) – Filename of the csv file. Defaults to `'nn_data.csv'`

Returns None (default) or DataFrame containing coordination data.

`surfaxe.analysis.`**`electrostatic_potential`**(*locpot='./LOCPOT'*, *lattice_vector=None*, *save_csv=True*, *csv_fname='potential.csv'*, *save_plt=True*, *plt_fname='potential.png'*, ***kwargs*)

Reads LOCPOT to get the planar and optionally macroscopic potential in c direction.

Parameters

- `locpot` (*str*, optional) – The path to the LOCPOT file. Defaults to `'./LOCPOT'`
- `lattice_vector` (*float*, optional) – The periodicity of the slab, calculates macroscopic potential with that periodicity
- `save_csv` (*bool*, optional) – Saves to csv. Defaults to `True`.
- `csv_fname` (*str*, optional) – Filename of the csv file. Defaults to `'potential.csv'`.
- `save_plt` (*bool*, optional) – Make and save the plot of electrostatic potential. Defaults to `True`.
- `plt_fname` (*str*, optional) – Filename of the plot. Defaults to `'potential.png'`.

Returns DataFrame

`surfaxe.analysis.`**`simple_nn`**(*start*, *end=None*, *ox_states=None*, *nn_method=<Mock name='mock()' id='140428077570128'>*, *save_csv=True*, *csv_fname='nn_data.csv'*)

Finds the nearest neighbours for simple structures. Before using on slabs make sure the nn_method works with the bulk structure.

The `site_index` in the produced DataFrame or csv file is one-indexed and represents the atom index in the structure.

Parameters

- `start` (*str*) – Filename of structure file in any format supported by pymatgen

- **end** (*str*, optional) – Filename of structure file in any format supported by pymatgen. Use if comparing initial and final structures. The structures must have same constituent atoms and number of sites. Defaults to `None`.
- **ox_states** (`None`, *list* or *dict*, optional) – Add oxidation states to the structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`
  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`
  - if `None`: The oxidation states are added by guess.

  Defaults to `None`.
- **nn_method** (*class*, optional) – The coordination number prediction algorithm used. Because the `nn_method` is a class, the class needs to be imported from pymatgen.analysis.local_env before it can be instantiated here. Defaults to `CrystalNN()`.
- **save_csv** (*bool*, optional) – Save to a csv file. Defaults to `True`.
- **csv_fname** (*str*, optional) – Filename of the csv file. Defaults to `'nn_data.csv'`

Returns None (default) or DataFrame containing coordination data

3.1.4 surfaxe.io module

`surfaxe.io.`**`plot_bond_analysis`**(*bond*, *df=None*, *filename=None*, *width=6*, *height=5*, *dpi=300*, *color=None*, *plt_fname='bond_analysis.png'*)

Plots the bond distance with respect to fractional coordinate. Used in conjunction with surfaxe.analysis.bond_analysis.

Parameters

- **bond** (*list*) – Bond to analyse; e.g. `['Y', 'O']` order of elements in the bond must be the same as in the Dataframe or provided file.
- **df** (*pandas DataFrame*, optional) – DataFrame from surfaxe.analysis.bond_analysis. Defaults to `None`.
- **filename** (*str*, optional) – Path to csv file with data from surfaxe.analysis.bond_analysis. Defaults to `None`. Either df or filename need to be supplied.
- **width** (*float*, optional) – Width of figure in inches. Defaults to `6`.
- **height** (*float*, optional) – Height of figure in inches. Defaults to `5`.
- **dpi** (*int*, optional) – Dots per inch. Defaults to `300`.
- **color** (*str*, optional) – Color of marker. Defaults to `None` which defaults to surfaxe base style
- **plt_fname** (*str*, optional) – Filename of the plot. Defaults to `'bond_analysis.png'`.

Returns None, saves plot to bond_analysis.png `surfaxe.io.`**`plot_electrostatic_potential`**(*df=None*, *filename=None*, *dpi=300*, *width=6*, *height=5*, *colors=None*, *plt_fname='potential.png'*)

Plots the planar and macroscopic electrostatic potential along one direction. Can take either a DataFrame or a potential.csv file as input.

Parameters

- **`df`** (*pandas DataFrame*, optional) – pandas DataFrame from surfaxe.analysis.electrostatic_potential. Defaults to `None`.
- **`filename`** (*str*, optional) – The filename of csv file with potential data. Defaults to `None`.
- **`dpi`** (*int*, optional) – Dots per inch. Defaults to 300.
- **`width`** (*float*, optional) – Width of figure in inches. Defaults to `6`.
- **`height`** (*float*, optional) – Height of figure in inches. Defaults to `5`.
- **`colors`** (*list*, optional) – A list of colours for planar and macroscopic potential plots. Defaults to `None`, which defaults to surfaxe base style.
- **`plt_fname`** (*str*, optional) – Filename of the plot. Defaults to `'potential.png'`.

Returns None, saves plot to potential.png

`surfaxe.io.`**`plot_enatom`**(*df*, *colors=None*, *dpi=300*, *width=6*, *height=5*, *plt_fname='energy_per_atom.png'*)

Plots the energy per atom for all terminations. Based on surfaxe.convergence parse_energies.

Parameters

- **`df`** (`pandas DataFrame`) – DataFrame from *parse_fols*, or any other Dataframe with headings 'slab_thickness, 'vac_thickness', 'slab_per_atom', 'time_taken', 'index'.
- **`colors`** (*list*, optional) – A list of colours for plots of different vacuum thicknesses. Defaults to `None`, which defaults to surfaxe base style.
- **`dpi`** (*int*, optional) – Dots per inch. Defaults to 300.
- **`width`** (*float*, optional) – Width of figure in inches. Defaults to `6`.
- **`height`** (*float*, optional) – Height of figure in inches. Defaults to `5`.
- **`plt_fname`** (*str*, optional) – The name of the plot. Defaults to `energy_per_atom.png`. If name with no format suffix is supplied, the format defaults to png.

Returns None, saves energy_per_atom.png

`surfaxe.io.`**`plot_surfen`**(*df*, *colors=None*, *dpi=300*, *width=8*, *height=8*, *plt_fname=None*)

Plots the surface energy for all terminations. Based on surfaxe.convergence parse_energies.

Parameters

- **`df`** (`pandas DataFrame`) – DataFrame from *parse_fols*, or any other Dataframe with headings 'slab_thickness', 'vac_thickness', 'surface_energy','surface_energy_boettger', 'surface_energy_fm', 'time_taken', 'index'.
- **`colors`** (*list*, optional) – A list of colours for plots of different surface energies. Defaults to `None`, which defaults to surfaxe base style.
- **`dpi`** (*int*, optional) – Dots per inch. Defaults to 300.
- **`width`** (*float*, optional) – Width of figure in inches. Defaults to `8`.
- **`height`** (*float*, optional) – Height of figure in inches. Defaults to `8`.

- **plt_fname** (*str*, optional) – The name of the plot. Defaults to `None` which is either `surface_energy.png` for one slab index or `surface_energy_slab_index.png` for multiple indices.

`surfaxe.io.`**`slab_from_file`**(*structure*, *hkl*)

Reads in structure from the file and returns slab object.

Parameters

- **structure** (`str`) – Structure file in any format supported by pymatgen. Will accept a pymatgen.Structure object directly.
- **hkl** (`tuple`) – Miller index of the slab in the input file.

Returns Slab object

`surfaxe.io.`**`slabs_to_file`**(*list_of_slabs*, *structure*, *make_fols*, *make_input_files*, *config_dict*, *fmt*, *name*, ***save_slabs_kwargs*)

Saves the slabs to file, optionally creates input files. The function can take any relevant keyword argument for DictSet.

Parameters

- **list_of_slabs** (*list*) – a list of slab dictionaries made with either of surfaxe.generation get_slab functions
- **structure** (*str*) – Filename of bulk structure file in any format supported by pymatgen.
- **make_fols** (*bool*) – Makes folders for each termination and slab/vacuum thickness combinations containing structure files.
  - `True`: A Miller index folder is created, in which folders named slab_vac_index are created to which the relevant structure files are saved.

    E.g. for a (0,0,1) slab of index 1 with a slab thickness of 20 Å and vacuum thickness of 30 Å the folder structure would be: `001/20_30_1/POSCAR`
  - `False`: The indexed structure files are put in a folder named after the bulk formula.

    E.g. for a (0,0,1) MgO slab of index 1 with a slab thickness of 20 Å and vacuum thickness of 30 Å the folder structure would be: `MgO/POSCAR_001_20_30_1.vasp`
- **make_input_files** (*bool*) – Makes INCAR, POTCAR and KPOINTS files in each folder. If `make_input_files` is `True` but `make_files` or `save_slabs` is `False`, files will be saved to folders regardless. This only works with VASP input files, other formats are not yet supported. Defaults to `False`.
- **config_dict** (*dict* or *str*) – Specifies the dictionary used for the generation of the input files.
- **fmt** (*str*, optional) – The format of the output files. Options include 'cif', 'poscar', 'cssr', 'json', not case sensitive. Defaults to 'poscar'.
- **name** (*str*, optional) – The name of the surface slab structure file created. Case sensitive. Defaults to 'POSCAR'

Returns None, saves surface slabs to file

3.1.5 surfaxe.vasp_data module

`surfaxe.vasp_data.`**`core_energy`**(*core_atom*, *bulk_nn*, *orbital='1s'*, *ox_states=None*, *nn_method=<Mock name='mock()' id='140428116002192'>*, *outcar='OUTCAR'*, *structure='POSCAR'*)

Parses the structure and OUTCAR files for the core level energy. Check the validity of nearest neighbour method on the bulk structure before using it on slabs.

Parameters

- **`core_atom`** (*str*, optional) – The symbol of atom the core state energy level should be parsed from.
- **`bulk_nn`** (*list*, optional) – The symbols of the nearest neighbours of the *core_atom*.
- **`orbital`** (*str*, optional) – The orbital of core state. Defaults to 1s.
- **`ox_states`** (`None`, *list* or *dict*, optional) – Add oxidation states to the structure. Different types of oxidation states specified will result in different pymatgen functions used. The options are:
  - if supplied as `list`: The oxidation states are added by site

    e.g. `[3, 2, 2, 1, -2, -2, -2, -2]`
  - if supplied as `dict`: The oxidation states are added by element

    e.g. `{'Fe': 3, 'O':-2}`
  - if `None`: The oxidation states are added by guess.

  Defaults to `None`.
- **`nn_method`** (*class* instance, optional) – The coordination number algorithm used. Because the `nn_method` is a class, the class needs to be imported from pymatgen.analysis.local_env before it can be instantiated here. Defaults to `CrystalNN()`.
- **`outcar`** (*str*, optional) – Path to the OUTCAR file. Defaults to `./OUTCAR`.
- **`structure`** (*str*, optional) – Path to the structure file in any format supported by pymatgen. Defaults to `./POSCAR`. Can also accept a pymaten.core.Structure object directly.

Returns Core state energy

`surfaxe.vasp_data.`**`process_data`**(*bulk_per_atom*, *parse_hkl=True*, *path_to_fols=None*, *hkl_dict=None*, *parse_core_energy=False*, *core_atom=None*, *bulk_nn=None*, *parse_vacuum=False*, *save_csv=True*, *csv_fname='data.csv'*, ***kwargs*)

Parses the folders to collect all final data on relevant input and output parameters, and optionally core and vacuum level energies.

If you are processing data for folder structures generated with *generation* make sure you use *convergence.parse_fols* function. This function is for parsing full sets of information from the output of production run calculations.

The folder structure for parsing of data is fairly flexible and can be: 1. automatically parsed if *parse_hkl=True* - the function searches for folders with names three digits long in cwd (default)

e.g. it finds folders cwd/100 and cwd/010 that correspond to Miller indices (1,0,0) and (0,1,0)

2. automatically parsed from a specific working directory if *path_to_fols* is specified 3. manually specified using *hkl_dict*, where the Miller index is mapped directly to the path to where the files are. If you are only interested in the specified folders, do not forget to change *parse_hkl=False*.

e.g. hkl_dict = {(0,1,1): 'path/to/001/files/', (2,0,1): 'path/to/201/files/'}

4. automatically parsed from cwd or a specific working directory in addition to a defined *hkl_dict*

Each of the folders must contain POSCAR and vasprun.xml files and optionally LOCPOT (or potential.csv) and OUTCAR files if vacuum or core energy are parsed.

The function returns None by default and saves the DataFrame to a csv file. Optionally, it can return the DataFrame.

Parameters

- **`bulk_per_atom`** (*float*) – Bulk energy per atom in eV per atom.
- **`parse_hkl`** (*bool*, optional) – If `True` the script parses the names of the folders to get the Miller indices. Defaults to `True`.
- **`path_to_fols`** (*str*, optional) – Path to where surfaxe should look for the hkl folders are. Defaults to None which searches in cwd.
- **`hkl_dict`** (*dict*, optional) – dictionary of tuples of Miller indices and paths to the folders the relevant outputs. Defaults to `None`. E.g. If the outputs of the calculations on the (1,-1,2) slab are in folder `path/to/folder/112`, the `hkl_dict` would be: {(1,-1,2): 'path/to/folder/112'}
- **`parse_core_energy`** (*bool*, optional) – If True the scripts attempts to parse core energies from a supplied OUTCAR. Defaults to `False`.
- **`core_atom`** (*str*, optional) – The symbol of atom the core state energy level should be parsed from. Defaults to `None`.
- **`bulk_nn`** (*list*, optional) – The symbols of the nearest neighbours of the *core_atom*. Defaults to `None`.
- **`parse_vacuum`** (*bool*, optional) – if `True` the script attempts to parse LOCPOT using analysis.electrostatic_potential to use the maximum value of planar potential as the vacuum energy level. Defaults to `True`.
- **`save_csv`** (*bool*, optional) – If `True`, it writes data to a csv file. Defaults to `True`.
- **`csv_fname`** (*str*, optional) – The filename of the csv. Defaults to data.csv

Returns DataFrame

`surfaxe.vasp_data.`**`vacuum`**(*path=None*)

Gets the energy of the vacuum level. It either parses potential.csv file if available or tries to calculate planar potential from LOCPOT. If neither file is available, function returns np.nan.

Parameters **`path`** (*str*, optional) – the path to potential.csv or LOCPOT files. Can be the path to a directory in which either file is or you can specify a path that must end in .csv or contain LOCPOT. Defaults to looking for potential.csv or LOCPOT in cwd.

Returns Maximum value of planar potential

CHAPTER

FOUR

TUTORIALS

We recommend starting off by looking at the dedicated tutorials. These Jupyter notebooks will guide you through most of the functionality of the package.

The tutorials can also be run interactively on Binder.

4.1 Using configuration dictionaries

One of the most powerful parts of surfaxe is its ability to make all VASP input files needed for convergence testing. To do so surfaxe makes use of configuration dictionaries (config dicts for short). These are python dictionaries that contain information used to set up INCAR, KPOINTS and POTCAR files.

For example, if we were interested in setting up a single shot PBEsol calculation on SnO2 slabs, we could set up the config dict as follows:

```
config_dict = {
"INCAR": {
    "ALGO": "Normal",
    "EDIFF": 1e-06,
    "EDIFFG": -0.01,
    "ENCUT": 500,
    "GGA": "PS",
    "ISMEAR": 0,
    "ISYM": 2,
    "IWAVPR": 1,
    "LASPH": true,
    "LORBIT": 11,
    "LREAL": "auto",
    "NELM": 200,
    "NSW": 0,
    "PREC": "Accurate",
    "SIGMA": 0.02
},
"KPOINTS": {
    "reciprocal_density": 55
},
"POTCAR": {
    "Sn": "Sn_d",
    "O" : "O"
}
}
```

Alternatively, one of the ready-made *surfaxe* config dicts (`PBEsol.json`, `PBEsol_relax.json`, `PBE.json`, `PBE_relax.json` or `HSE06.json`) can be used and further modified using `user_incar_settings`, `user_kpoints_settings` and `user_potcar_settings`. The `relax` config dicts contain additional parameters necessary for geometric relaxations of slabs. The POTCAR functional (i.e. PBE, PBE_54) can be chosen with `user_potcar_functional`.

Pymatgen documentation covers exact behaviour of the `user_incar_settings`, `user_kpoints_settings` and `user_potcar_settings` and all additional keyword arguments that can be supplied to slab generation scripts.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX