site$_{a}nalysisDocumentation$

Release 0.0.1

Benjamin J. Morgan

Oct 16, 2022

Contents:

CHAPTER 1

Installation

`site-analysis` can be installed from the PyPI package manager with `pip`:

```
pip install site-analysis
```

Alternatively, the latest development build can be found Github.

CHAPTER 2

Context

The diffusion of ions through solid materials is a fundamental physical process that underpins applications such as lithium-ion batteries, fuel cells, and chemical sensors. One common technique for studying atomic scale diffusion processes in solids is molecular dynamics simulation, which can be used to directly calculate macroscopic transport coefficients, i.e. diffusion coefficients and ionic conductivities, and also to examine the detailed atomic scale mechanisms of ionic transport. While molecular dynamics simulations, in principle, provides a description of all atomic scale dynamical processes that occur on a simulation timescale, analysing the resulting raw data to extract quantitative data about diffusion mechanisms can be challenging.

One approach to obtain mechanistic information from a molecular dynamics simulation is to spatially discretise the atomic trajectories by projecting from a set of three dimensional continuous coordinates onto a set of discrete "sites". This approach is founded on the idea that in many solids, diffusion proceeds by ions undergoing a sequence of "jumps" between local potential energy minima. These minima typically correspond to a particular arrangement of the mobile ions within some set of crystallographic sites. In a site-projection analysis, we first define a set of bounded volumes within the simulation cell, which represent our "sites", and then project the coordinates of the mobile ions onto these volumes. This gives two spatially discretised representations of each configuration in a simulation trajectory. From the perspective of the atoms, each atom is assigned to zero, one, or multiple sites (depending on whether out sites are defined to be mutually space-filling and / or non-overlapping). From the perspective of the sites, each site is occupied by zero, one, or more mobile ions.

This site occupation data can be used to build quantiative descriptions of diffusion mechanisms. For example, we can calculate time-averaged site-occupation probabilities, which can be compared to experimental results, such as diffraction data. Because the site-occupation data is time-resolved, we can also use this to analyse the trajectories of mobile atoms. For example, specific sequences of sites that an ion moves through can be statistically analysed, or temporal and spatical correlations between the movements of groups of mobile atoms can be quantified.

CHAPTER 3

2D lattice examples

To illustrate the concepts and techniques used in `site_analysis` consider a periodic 4×4 square lattice, which defines a set of 16 sites. We then introduce a single mobile atom. For each configuration we are interested in assigning the mobile ion to one of the 16 sites.

TODO: - foo

[ ]:

CHAPTER 4

Tutorials

[1]:
```
import numpy as np
```

4.1 VoronoiSite example trajectory analysis

For this example, we want to analyse a simulation trajectory using voronoi sites.

We can define our sites by creating a series of `pymatgen Structures`, using the `Structure.from_spacegroup()` method. Each structure contains only Na sites, using the coordinates from Ramos *et al. Chem. Mater.* 2018.

[6]:
```
from pymatgen.symmetry.groups import SpaceGroup
from pymatgen.io.vasp import Poscar
sg = SpaceGroup('I41/acd:2')
all_na_structure = Poscar.from_file('./na_sn_all_na_new.POSCAR.vasp').structure
```

[7]:
```
from pymatgen import Structure, Lattice
lattice = all_na_structure.lattice
na1 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.25, 0.0, 0.125]])
na2 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.00, 0.0, 0.125]])
na3 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.0, 0.25, 0.0]])
na4 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.0, 0.0, 0.0]])
na5 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.75, 0.25, 0.0]])
na6 = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.5, 0.75, 0.625]])
i2  = Structure.from_spacegroup(sg='I41/acd:2', lattice=lattice, species=['Na'],␣
↪coords=[[0.666, 0.1376, 0.05]])
```

(continues on next page)

(continued from previous page)

```
na_structures = {'Na1': na1,
                 'Na2': na2,
                 'Na3': na3,
                 'Na4': na4,
                 'Na5': na5,
                 'Na6': na6,
                 'i2': i2}
```

[8]:
```
from site_analysis.voronoi_site import VoronoiSite
na1_sites = [VoronoiSite(s.frac_coords, label='Na1') for s in na1 ]
na2_sites = [VoronoiSite(s.frac_coords, label='Na2') for s in na2 ]
na3_sites = [VoronoiSite(s.frac_coords, label='Na3') for s in na3 ]
na4_sites = [VoronoiSite(s.frac_coords, label='Na4') for s in na4 ]
na5_sites = [VoronoiSite(s.frac_coords, label='Na5') for s in na5 ]
na6_sites = [VoronoiSite(s.frac_coords, label='Na6') for s in na6 ]
i2_sites  = [VoronoiSite(s.frac_coords, label='i2') for s in i2 ]
sites = na1_sites + na2_sites + na3_sites + na4_sites + na5_sites + na6_sites + i2_
↪sites
```

[9]:
```
from pymatgen.io.vasp import Poscar
structure = Poscar.from_file('POSCAR').structure
print(structure.composition)
# create Atom objects
from site_analysis.atom import atoms_from_species_string
atoms = atoms_from_species_string(structure, 'Na')
atoms[0:3]
```

```
Na88 Sn16 P8 S96
```

[9]:
```
[site_analysis.Atom(index=0, in_site=None, frac_coords=None),
 site_analysis.Atom(index=1, in_site=None, frac_coords=None),
 site_analysis.Atom(index=2, in_site=None, frac_coords=None)]
```

We now create a `Trajectory` object

[10]:
```
from site_analysis.trajectory import Trajectory
trajectory = Trajectory(sites=sites, atoms=atoms)
trajectory
```

[10]:
```
<site_analysis.trajectory.Trajectory at 0x12a322dd8>
```

To analyse the site occupation for a particular `pymatgen Structure`:

[11]:
```
trajectory.analyse_structure(structure)
```

[12]:
```
np.array(trajectory.atom_sites)
```

[12]:
```
array([16, 32, 36, 20,  3, 38, 34, 23,  5, 33, 35, 19, 37, 21, 24, 12, 28,
       46, 30, 42, 26, 47, 31, 25, 41, 43, 27, 29, 50, 54, 79, 53, 49, 51,
       64, 58, 62, 61, 63, 59, 91, 83, 89, 81, 90, 82, 52, 80, 95, 87, 57,
       93, 86, 94, 92, 84, 48, 76, 78, 68, 70, 60, 77, 66, 55, 75, 65, 72,
       73, 67, 74, 69, 13, 18,  4, 17, 14,  6,  7, 15,  9,  1,  8,  0,  2,
       44, 45, 11])
```

[13]:
```
from pymatgen.io.vasp import Xdatcar
trajectory.reset()
xdatcar = Xdatcar('XDATCAR')
```

(continues on next page)

(continued from previous page)

```
for timestep, s in enumerate(xdatcar.structures):
    trajectory.append_timestep(s, t=timestep)
```

Rough example for collecting only occupied sites, and counting their site types:

[14]:
```
from collections import Counter
c = Counter()
for site in trajectory.sites:
    c[site.label] += len([ 1 for ts in site.trajectory if len(ts)>0 ])
for k, v in c.items():
    print( k, v/len(trajectory))
```

```
Na1 15.35
Na2 29.75
Na3 12.4
Na4 15.15
Na5 15.15
Na6 0.0
i2 0.2
```

vs. all sites:

[15]:
```
c_sites = Counter(trajectory.site_labels())
c_sites
```

[15]:
```
Counter({'Na1': 16,
         'Na2': 32,
         'Na3': 16,
         'Na4': 16,
         'Na5': 16,
         'Na6': 8,
         'i2': 32})
```

[17]:
```
trajectory.reset()

xdatcar = Xdatcar('XDATCAR_Sn')

trajectory.trajectory_from_structures( xdatcar.structures, progress='notebook')
```

```
HBox(children=(IntProgress(value=0, max=300), HTML(value='')))
```

[19]:
```
n_timesteps = len(trajectory.timesteps)
c_sites = Counter(trajectory.site_labels())
c = Counter()
p_occ = {}
for site in trajectory.sites:
    c[site.label] += len([ 1 for ts in site.trajectory if len(ts)>0 ])
for k, v in c.items():
    p_occ[k] = v / c_sites[k] / n_timesteps
p_occ
```

[19]:
```
{'Na1': 0.93375,
 'Na2': 0.895625,
 'Na3': 0.915,
 'Na4': 0.9427083333333334,
 'Na5': 0.9120833333333334,
```

(continues on next page)

(continued from previous page)

```
 'Na6': 0.0,
 'i2': 0.0026041666666666665}
```

[20]:
```
# check total average occupation = 88 atoms
for k,v in c.items():
    print( k, p_occ[k]*c_sites[k])
print( sum( [ p_occ[k] * c_sites[k] for k, v in c.items()]))
```

```
Na1 14.94
Na2 28.66
Na3 14.64
Na4 15.083333333333334
Na5 14.593333333333334
Na6 0.0
i2 0.08333333333333333
88.0
```

[34]:
```
def residence_times(atom_trajectory):
    """Calculates the numbers of sequential timesteps when an atom
    occupies the same site.

    Args:
        atom_trajectory (list): List of site indices.

    Returns:
        (dict)

    Example:
        >>> atom_trajectory = [3, 3, 3, 7, 7, 5, 3]
        >>> residence_times(atom_trajectory)
        {3: [3, 1], 7: [2], 5: [1]}

    """
    r_times = {}
    current_site = None
    for site in atom_trajectory:
        if site != current_site:
            if site in r_times:
                r_times[site].append(1)
            else:
                r_times[site] = [1]
        else:
            r_times[site][-1] += 1
        current_site = site
    return r_times
```

[35]:
```
residence_times([3, 3, 3, 7, 7, 5, 3])
```

[35]:
```
{3: [3, 1], 7: [2], 5: [1]}
```

[36]:
```
residence_times(trajectory.atoms[0].trajectory)
```

[36]:
```
{16: [300]}
```

[44]:
```
# Look at the site trajectory of atom 0:
r_times = []
```

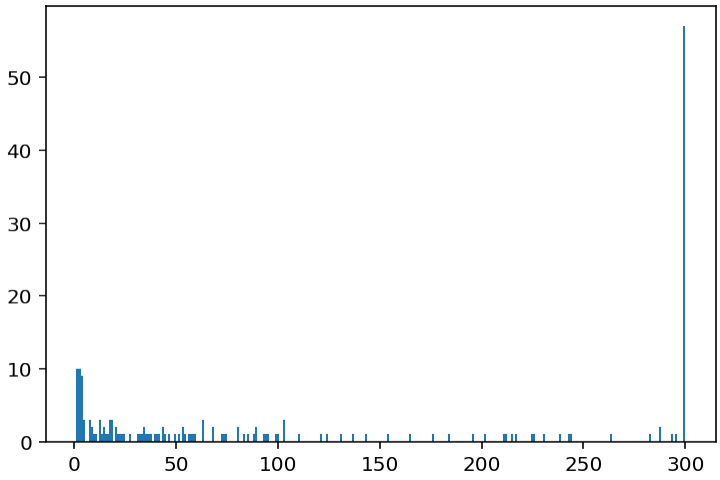
(continues on next page)

(continued from previous page)

```
for a in trajectory.atoms:
    rt = residence_times(a.trajectory)
    for k, v in rt.items():
        r_times.extend(v)
```

[46]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[49]:
```
plt.hist(r_times, bins=300)
plt.show()
```



[38]:
```
def smooth_trajectory(t):
    for i in range(len(t)-2):
        if t[i] == t[i+2]:
            t[i+1] = t[i]
    return t
```

[41]:
```
t = [0,0,0,1,0,1,1]
smooth_trajectory(t)
```

[41]:
```
[0, 0, 0, 0, 0, 1, 1]
```

[42]:
```
for a in trajectory.atoms:
    print(residence_times(smooth_trajectory(a.trajectory)))
```

```
{16: [300]}
{32: [300]}
{36: [296], 83: [4]}
{20: [300]}
{3: [110, 121], 22: [68], 112: [1]}
{38: [300]}
{34: [300]}
{23: [300]}
{5: [300]}
{33: [300]}
{35: [300]}
{19: [300]}
{37: [244], 7: [56]}
```

(continues on next page)

(continued from previous page)

```
{21: [300]}
{24: [131, 103], 3: [3], 95: [63]}
{12: [184, 94], 40: [22]}
{28: [100, 15, 17, 80], 95: [18, 34, 20, 2, 7], 131: [3, 1, 1, 2]}
{46: [300]}
{30: [300]}
{42: [300]}
{26: [300]}
{47: [300]}
{31: [300]}
{25: [226], 8: [74]}
{41: [243], 84: [57]}
{43: [300]}
{10: [202, 95], 27: [3]}
{29: [300]}
{50: [300]}
{54: [300]}
{79: [300]}
{53: [283], 89: [17]}
{49: [300]}
{51: [300]}
{64: [31, 93], 56: [176]}
{58: [300]}
{62: [300]}
{61: [300]}
{63: [300]}
{59: [300]}
{91: [300]}
{83: [288], 53: [12]}
{89: [196], 121: [1], 22: [103]}
{81: [300]}
{90: [300]}
{82: [300]}
{52: [68, 53, 85], 88: [53, 40], 120: [1]}
{80: [300]}
{95: [23, 58], 57: [2, 217]}
{87: [59, 239], 117: [1], 40: [1]}
{57: [12], 85: [288]}
{93: [300]}
{86: [300]}
{94: [99, 137], 27: [63], 132: [1]}
{92: [300]}
{84: [212], 56: [88]}
{48: [73, 103], 64: [124]}
{76: [300]}
{78: [300]}
{68: [72, 8, 215], 52: [2, 3]}
{70: [300]}
{60: [300]}
{77: [89, 20, 154], 48: [35, 2]}
{66: [300]}
{55: [27, 14, 32, 3, 8, 18, 2, 3, 39], 71: [51, 4, 10, 3, 13, 43, 18, 12]}
{75: [300]}
{65: [300]}
{72: [36, 16, 17, 225], 57: [2, 2, 2]}
{73: [300]}
{67: [300]}
```

(continues on next page)

(continued from previous page)

```
{74: [46, 83, 143], 55: [24, 4]}
{69: [300]}
{13: [300]}
{18: [231, 54], 127: [1], 89: [14]}
{4: [3, 294], 22: [3]}
{17: [33, 165, 43], 88: [21, 37], 126: [1]}
{14: [300]}
{6: [300]}
{7: [211], 27: [89]}
{15: [300]}
{9: [300]}
{1: [300]}
{8: [49, 7, 7, 9], 39: [63, 44, 41, 80]}
{0: [300]}
{2: [300]}
{44: [300]}
{45: [300]}
{11: [2, 264], 40: [34]}
```

[ ]:

CHAPTER 5

API documentation

5.1 Submodules

5.1.1 site_analysis.atom

class Atom(*index: int*, *species_string: Optional[str] = None*)

Bases: `object`

Represents a single persistent atom during a simulation.

index

Unique numeric index identifying this atom.

Type int

in_site

Site index for the site this atom currently occupies.

Type int

frac_coords

Numpy array containing the current fractional coordinates for this atom.

Type np.array

trajectory

List of site indices occupied at each timestep.

Type list

Note: The atom index is used to identify it when parsing structures, so needs to be e.g. the corresponding Site index in a Pymatgen Structure.

---

as_dict() → Dict[KT, VT]

assign_coords(*structure: pymatgen.core.structure.Structure*) → None
Assign fractional coordinates to this atom from a pymatgen Structure.

Parameters **structure** (*pymatgen.Structure*) – The Structure to use for this atom's fractional coordinates.

Returns None

frac_coords
Getter for the fractional coordinates of this atom.

Raises AttributeError – if the fractional coordinates for this atom have not been set.

classmethod from_dict(*d: Dict[KT, VT]*) → site_analysis.atom.Atom

classmethod from_file(*filename*)

classmethod from_str(*input_string: str*) → site_analysis.atom.Atom
Initiate an Atom object from a JSON-formatted string.

Parameters **input_string** (*str*) – JSON-formatted string.

Returns (Atom)

reset() → None
Reset the state of this Atom.

Clears the *in_site* and *trajectory* attributes.

Returns None

to(*filename: Optional[str] = None*) → str

atoms_from_indices(*indices: List[int]*) → List[site_analysis.atom.Atom]

atoms_from_species_string(*structure: pymatgen.core.structure.Structure*, *species_string: str*) → List[site_analysis.atom.Atom]

5.1.2 site_analysis.polyhedral_site

class PolyhedralSite(*vertex_indices: List[int], label: Optional[str] = None*)
Bases: *site_analysis.site.Site*

Describes a site defined by the polyhedral volume enclosed by a set of vertex atoms.

index
Numerical ID, intended to be unique to each site.

Type int

label (`str`
optional): Optional string given as a label for this site. Default is *None*.

contains_atoms
List of the atoms contained by this site in the structure last processed.

Type list

trajectory
List of sites this atom has visited at each timestep?

Type list

points
List of fractional coordinates for atoms assigned as occupying this site.

Type list

transitions
Stores observed transitions from this site to other sites. Format is {index: count} with `index` giving the index of each destination site, and `count` giving the number of observed transitions to this site.

Type collections.Counter

vertex_indices
List of integer indices for the vertex atoms (counting from 0).

Type list(int)

label
Optional label for the site.

Type `str`, optional

as_dict() → Dict[KT, VT]
Json-serializable dict representation of this Site.

Parameters **None** –

Returns (dict)

assign_vertex_coords(*structure: pymatgen.core.structure.Structure*) → None
Assign fractional coordinates to the polyhedra vertices from the corresponding atom positions in a pymatgen Structure.

Parameters **structure** (*Structure*) – The pymatgen Structure used to assign the vertices fractional coordinates.

Returns None

Notes

This method assumes the coordinates of the vertices may have changed, so unsets the Delaunay tesselation for this site.

centre() → numpy.ndarray
Returns the fractional coordinates of the centre point of this polyhedral site.

Parameters **None** –

Returns (3,) numpy array.

Return type (np.array)

cn
Coordination number for this site, defined as the number of vertices

Convenience property for coordination_number()

Returns int

contains_atom(*atom: site_analysis.atom.Atom*, *algo: Optional[str] = 'simplex'*, **args*, ***kwargs*)
→ bool
Test whether an atom is inside this polyhedron.

Parameters

- **atom** (Atom) – The atom to test.
- **algo** (`str`, optional) – Select the algorithm to us. Options are 'simplex' and 'sn'. See the documentation for the contains_point() method for more details. Default is 'simplex'.

Returns bool

contains_point (*x: numpy.ndarray*, *structure: Optional[pymatgen.core.structure.Structure] = None*, *algo: str = 'simplex'*, **args*, ***kwargs*) → bool

Test whether a specific point is enclosed by this polyhedral site.

Parameters

- **x** (*np.array*) – Fractional coordinates of the point to test (length 3 numpy array).
- **structure** (Structure, optional) – Optional pymatgen Structure. If provided, the vertex coordinates for this polyhedral site will be assigned using this structure. Default is None.
- **algo** (*str*) – Select the algorithm for testing whether a point is contained by the site:

  simplex: Use scipy.spatial.Delaunay.find_simplex to test if any of the simplices that make up this polyhedron contain the point.

  sn: Compute the sign of the surface normal for each polyhedron face with respect to the point, to test if the point lies "inside" every face.

Returns bool

contains_point_simplex (*x: numpy.ndarray*) → bool

Test whether one or more points are inside this site, by checking whether these points are contained inside the simplices of the Delaunay tesselation defined by the vertex coordinates.

Parameters **x** (*np.array*) – Fractional coordinates for one or more points, as a (3x1) or (3xN) numpy array.

Returns bool

contains_point_sn (*x_list: numpy.ndarray*) → bool

Test whether one or more points are inside this site, by calculating the sign of the surface normal for each face with respect to each point.

Parameters **x** (*np.array*) – Fractional coordinates for one or more points, as a (3x1) or (3xN) numpy array.

Returns bool

Note: This method could be made more efficient by caching the surface_normal vectors and in-face vectors.

This is also a possible target for optimisation with f2py etc.

---

coordination_number

Coordination number for this site, defined as the number of vertices

Returns int

delaunay

Delaunay tessellation of the vertex coordinates for this site.

This is calculated the first time the attribute is requested, and then stored for reuse, unless the site is reset.

Returns scipy.spatial.Delaunay

classmethod from_dict (*d*)

Create a Site object from a dict representation.

Parameters **d** (*dict*) – The dict representation of this Site.

Returns (Site)

get_vertex_species (*structure: pymatgen.core.structure.Structure*) → List[str]
Returns a list of species strings for the vertex atoms of this polyhedral site.

Parameters **structure** (*Structure*) – Pymatgen Structure used to assign species to each vertex atom.

Returns List of species strings of the vertex atoms.

Return type (list(str))

reset () → None
Reset the trajectory for this site.

Resets the contains_atoms and trajectory attributes to empty lists.

Vertex coordinates and Delaunay tesselation are unset.

Parameters **None** –

Returns None

classmethod sites_from_vertex_indices (*vertex_indices*, *label=None*)

5.1.3 site_analysis.polyhedral_site_collection

class PolyhedralSiteCollection (*sites: List[site_analysis.site.Site]*)
Bases: *site_analysis.site_collection.SiteCollection*

A collection of PolyhedralSite objects.

sites
List of Site-like objects.

Type list

analyse_structure (*atoms: List[site_analysis.atom.Atom], structure: pymatgen.core.structure.Structure*)
Perform a site analysis for a set of atoms on a specific structure.

This method should be implemented in the derived subclass.

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

assign_site_occupations (*atoms: List[site_analysis.atom.Atom], structure: pymatgen.core.structure.Structure*)
Assigns atoms to sites for a specific structure.

This method should be implemented in the derived subclass

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

Notes

The atom coordinates should already be consistent with the coordinates in *structure*. Recommended usage is via the `analyse_structure()` method.

neighbouring_sites(*index: int*) → List[site_analysis.polyhedral_site.PolyhedralSite]
If implemented, returns a list of sites that neighbour a given site.

This method should be implemented in the derived subclass.

Parameters **site_index** (*int*) – Index of the site to return a list of neighbours for.

sites_contain_points(*points: numpy.ndarray*, *structure: Optional[pymatgen.core.structure.Structure] = None*) → bool
Checks whether the set of sites contain a corresponding set of fractional coordinates.

Parameters

- **points** (*np.array*) – 3xN numpy array of fractional coordinates. There should be one coordinate for each site being checked.
- **structure** (*Structure*) – Pymatgen Structure used to define the vertex coordinates of each polyhedral site.

Returns (bool)

construct_neighbouring_sites(*sites: List[site_analysis.polyhedral_site.PolyhedralSite]*) → Dict[int, List[site_analysis.polyhedral_site.PolyhedralSite]]
Find all polyhedral sites that are face-sharing neighbours.

Any polyhedral sites that share 3 or more vertices are considered to share a face.

Parameters **None** –

Returns

Dictionary of *int*: *list* entries. Keys are site indices. Values are lists of `PolyhedralSite` objects.

Return type (dict)

5.1.4 site_analysis.site

class Site(*label: Optional[str] = None*)
Bases: `abc.ABC`

Parent class for defining sites.

A Site is a bounded volume that can contain none, one, or more atoms. This class defines the attributes and methods expected for specific Site subclasses.

index
Numerical ID, intended to be unique to each site.

Type int

label (`str`
optional): Optional string given as a label for this site. Default is *None*.

contains_atoms
List of the atoms contained by this site in the structure last processed.

Type list

trajectory
Nested list of atoms that have visited this site at each timestep.

Type list(list(int))

points
List of fractional coordinates for atoms assigned as occupying this site.

Type list

transitions
Stores observed transitions from this site to other sites. Format is {index: count} with `index` giving the index of each destination site, and `count` giving the number of observed transitions to this site.

Type collections.Counter

as_dict() → Dict[KT, VT]
Json-serializable dict representation of this Site.

Parameters **None** –

Returns (dict)

centre() → numpy.ndarray
Returns the centre point of this site.

This method should be implemented in the derived subclass.

Parameters **None** –

Returns None

contains_atom(*atom: site_analysis.atom.Atom*, **args*, ***kwargs*) → bool
Test whether this site contains a specific atom.

Parameters **atom** (Atom) – The atom to test.

Returns (bool)

contains_point(*x: numpy.ndarray*, **args*, ***kwargs*) → bool
Test whether the fractional coordinate x is contained by this site.

This method should be implemented in the derived subclass

Parameters **x** (*np.array*) – Fractional coordinate.

Returns (bool)

Note: Specific Site subclasses may require additional arguments to be passed.

---

coordination_number() → int
Returns the coordination number of this site.

This method should be implemented in the derived subclass.

Parameters **None** –

Returns int

classmethod from_dict(*d: Dict[KT, VT]*) → site_analysis.site.Site
Create a Site object from a dict representation.

Parameters **d** (*dict*) – The dict representation of this Site.

Returns (Site)

reset() → None
Reset the trajectory for this site.

Returns the contains_atoms and trajectory attributes to empty lists.

Parameters **None** –

Returns None

classmethod reset_index(*newid: int = 0*) → None
Reset the site index counter.

Parameters **(int** (*newid*) – optional): New starting index. Default is 1.

Returns None

5.1.5 site_analysis.site_collection

class SiteCollection(*sites: Sequence[site_analysis.site.Site]*)
Bases: abc.ABC

Parent class for collections of sites.

Collections of specific site types should inherit from this class.

sites
List of Site-like objects.

Type list

analyse_structure(*atoms*, *structure*)
Perform a site analysis for a set of atoms on a specific structure.

This method should be implemented in the derived subclass.

Parameters

- **atoms** (*list (Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

assign_site_occupations(*atoms*, *structure*)
Assigns atoms to sites for a specific structure.

This method should be implemented in the derived subclass

Parameters

- **atoms** (*list (Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

Notes

The atom coordinates should already be consistent with the coordinates in *structure*. Recommended usage is via the analyse_structure() method.

neighbouring_sites(*site_index*)
If implemented, returns a list of sites that neighbour a given site.

This method should be implemented in the derived subclass.

Parameters **site_index** (*int*) – Index of the site to return a list of neighbours for.

reset_site_occupations()
Occupations of all sites in this site collection are set as empty.

Parameters **None** –

Returns None

site_by_index(*index*)
Returns the site with a specific index.

Parameters **index** (*int*) – index for the site to be returned.

Returns (Site)

Raises ValueError – If a site with the specified index is not contained in this SiteCollection.

sites_contain_points(*points: numpy.ndarray*, *structure: Optional[pymatgen.core.structure.Structure] = None*) → bool
If implemented, Checks whether the set of sites contain a corresponding set of fractional coordinates. :param points: 3xN numpy array of fractional coordinates.

There should be one coordinate for each site being checked.

Returns (bool)

Notes

Specific SiteCollection subclass implementations may require additional arguments to be passed.

update_occupation(*site*, *atom*)
Updates site and atom attributes for this atom occupying this site.

Parameters

- **site** (Site) – The site to be updated.
- **atom** (Atom) – The atom to be updated.

Returns None

Notes

This method does the following:

1. If the atom has moved to a new site, record a old_site –> new_site transition.
2. Add this atom's index to the list of atoms occupying this site.
3. Add this atom's fractional coordinates to the list of coordinates observed occupying this site.
4. Assign this atom this site index.

5.1.6 site_analysis.spherical_site

class SphericalSite(*frac_coords: numpy.ndarray*, *rcut: float*, *label: Optional[str] = None*)
Bases: *site_analysis.site.Site*

as_dict() → Dict[KT, VT]
Json-serializable dict representation of this Site.

Parameters **None** –

Returns (dict)

centre() → numpy.ndarray
Returns the centre point of this site.

This method should be implemented in the derived subclass.

Parameters **None** –

Returns None

contains_atom(*atom: site_analysis.atom.Atom*, *lattice: Optional[pymatgen.core.lattice.Lattice] = None*, **args*, ***kwargs*) → bool
Test whether this site contains a specific atom.

Parameters **atom** (Atom) – The atom to test.

Returns (bool)

contains_point(*x: numpy.ndarray*, *lattice: Optional[pymatgen.core.lattice.Lattice] = None*, **args*, ***kwargs*) → bool
Test whether the fractional coordinate x is contained by this site.

This method should be implemented in the derived subclass

Parameters **x** (*np.array*) – Fractional coordinate.

Returns (bool)

Note: Specific Site subclasses may require additional arguments to be passed.

---

classmethod from_dict(*d: Dict[KT, VT]*) → site_analysis.spherical_site.SphericalSite
Create a Site object from a dict representation.

Parameters **d** (*dict*) – The dict representation of this Site.

Returns (Site)

5.1.7 site_analysis.spherical_site_collection

class SphericalSiteCollection(*sites: Sequence[site_analysis.site.Site]*)
Bases: *site_analysis.site_collection.SiteCollection*

analyse_structure(*atoms: List[site_analysis.atom.Atom]*, *structure: pymatgen.core.structure.Structure*) → None
Perform a site analysis for a set of atoms on a specific structure.

This method should be implemented in the derived subclass.

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.

- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

assign_site_occupations(*atoms: List[site_analysis.atom.Atom], structure: pymatgen.core.structure.Structure*) → None

Assigns atoms to sites for a specific structure.

This method should be implemented in the derived subclass

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

Notes

The atom coordinates should already be consistent with the coordinates in *structure*. Recommended usage is via the `analyse_structure()` method.

5.1.8 site_analysis.tools

site_analysis.tools module

This module contains tools for [TODO]

get_nearest_neighbour_indices(*structure: pymatgen.core.structure.Structure, ref_structure: pymatgen.core.structure.Structure, vertex_species: List[str], n_coord: int*) → List[List[int]]

Returns the atom indices for the N nearest neighbours to each site in a reference structure.

Parameters

- **structure** (*pymatgen.Structure*) – A pymatgen Structure object, used to select the nearest neighbour indices.
- **ref_structure** (*pymatgen.Structure*) – A pymatgen Structure object. Each site is used to find the set of N nearest neighbours (of the specified atomic species) in `structure`.
- **vertex_species** (*list(str)*) – List of strings specifying the atomic species of the vertex atoms, e.g. `[ 'S', 'I' ]`.
- **n_coord** (*int*) – Number of matching nearest neighbours to return for each site in `ref_structure`.

Returns N_sites x N_neighbours nested list of vertex atom indices.

Return type (list(list(int))

get_vertex_indices(*structure: pymatgen.core.structure.Structure, centre_species: str, vertex_species: Union[str, List[str]], cutoff: float = 4.5, n_vertices: Union[int, List[int]] = 6*) → List[List[int]]

Find the atom indices for atoms defining the vertices of coordination polyhedra, from a pymatgen Structure object.

Given the elemental species of a set of central atoms, A, and of the polyhedral vertices, B, this function finds: for each A, then N closest neighbours B (within some cutoff). The number of neighbours found per central atom can be a single value for all A, or can be provided as a list of values for each A.

Parameters

- **structure** (*pymatgen.Structure*) – A pymatgen Structure object, used to find the coordination polyhedra vertices..
- **centre_species** (*str*) – Species string identifying the atoms at the centres of each coordination environment, e.g. "Na".
- **vertex_species** (*str or list(str)*) – Species string identifying the atoms at the vertices of each coordination environment, e.g. "S"., or a list of strings, e.g. `["S", "I"]`.
- **cutoff** (*float*) – Distance cutoff for neighbour search.
- **n_vertices** (*int or list(int)*) – Number(s) of nearest neighbours to return for each set of vertices. If a list is passed, this should be the same length as the number of atoms of centre species A.

Returns

Nested list of integers, giving the atom indices for each coordination environment.

Return type list(list(int))

site_index_mapping(*structure1: pymatgen.core.structure.Structure, structure2: pymatgen.core.structure.Structure, species1: Union[str, List[str], None] = None, species2: Union[str, List[str], None] = None, one_to_one_mapping: Optional[bool] = True, return_mapping_distances: Optional[bool] = False*) → Union[numpy.ndarray, Tuple[numpy.ndarray, numpy.ndarray]]

Compute the site index mapping between two structures based on the closest corresponding site in structure2 to each selected site in structure1.

Parameters

- **structure1** (*pymatgen.Structure*) – The structure to map from.
- **structure2** (*pymatgen.Structure*) – The structure to map to.
- **species1** (*optional, str or list(str)*) – Optional argument to select a subset of atomic species to map site indices from.
- **species2** (*optional, str of list(str)*) – Optional argument to specify a subset of atomic species to map site indices to.
- **one_to_one_mapping** (*optional, bool*) – Optional argument to check that a one-to-one mapping is found between the relevant subsets of sites in structure1 and structure2. Default is *True*.

Returns np.ndarray

Raises `ValueError` – if *one_to_one_mapping* = *True* and a one-to-one mapping is not found.

species_string_from_site(*site: pymatgen.core.sites.Site*)

x_pbc(*x: numpy.ndarray*)

Return an array of fractional coordinates mapped into all positive neighbouring periodic cells.

Parameters **x** (*np.array*) – Input fractional coordinates.

Returns

(9,3) numpy array of all mapped fractional coordinates, including the original coordinates in the origin calculation cell.

Return type np.array

Example

```
>>> x = np.array([0.1, 0.2, 0.3])
>>> x_pbc(x)
array([[0.1, 0.2, 0.3],
       [1.1, 0.2, 0.3],
       [0.1, 1.2, 0.3],
       [0.1, 0.2, 1.3],
       [1.1, 1.2, 0.3],
       [1.1, 0.2, 1.3],
       [0.1, 1.2, 1.3],
       [1.1, 1.2, 1.3]])
```

5.1.9 site_analysis.trajectory

class Trajectory(*sites: List[site_analysis.site.Site], atoms: List[site_analysis.atom.Atom]*)

Bases: object

Class for performing sites analysis on simulation trajectories.

analyse_structure(*structure: pymatgen.core.structure.Structure*) → None

append_timestep(*structure: pymatgen.core.structure.Structure*, *t: Optional[int] = None*) → None

assign_site_occupations(*structure: pymatgen.core.structure.Structure*) → None

at

atom_by_index(*i: int*) → site_analysis.atom.Atom

atom_sites

atoms_trajectory

reset() → None

site_by_index(*i: int*) → site_analysis.site.Site

site_coordination_numbers() → collections.Counter

site_labels() → List[Optional[str]]

site_occupations

sites_trajectory

st

trajectory_from_structures(*structures*, *progress=False*)

update_occupation(*site*, *atom*)

5.1.10 site_analysis.voronoi_site

class VoronoiSite(*frac_coords: numpy.ndarray*, *label: str = None*)

Bases: *site_analysis.site.Site*

Site subclass corresponding to Voronoi cells centered on fixed positions.

frac_coords
Fractional coordinates of the site centre.

Type np.array

as_dict() → Dict[KT, VT]
Json-serializable dict representation of this VoronoiSite.

Parameters **None** –

Returns (dict)

centre() → numpy.ndarray
Returns the centre position of this site.

Parameters **None** –

Returns np.ndarray

contains_point(*x: numpy.ndarray*, **args*, ***kwargs*) → bool
A single Voronoi site cannot determine whether it contains a given point, because the site boundaries are defined by the set of all Voronoi sites.

Use VoronoiSiteCollection.assign_site_occupations() instead.

classmethod from_dict(*d*)
Create a VoronoiSite object from a dict representation.

Parameters **d** (*dict*) – The dict representation of this Site.

Returns (VoronoiSite)

5.1.11 site_analysis.voronoi_site_collection

class VoronoiSiteCollection(*sites: List[site_analysis.site.Site]*)
Bases: *site_analysis.site_collection.SiteCollection*

analyse_structure(*atoms: List[site_analysis.atom.Atom]*, *structure: pymatgen.core.structure.Structure*) → None
Perform a site analysis for a set of atoms on a specific structure.

This method should be implemented in the derived subclass.

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

assign_site_occupations(*atoms: List[site_analysis.atom.Atom]*, *structure: pymatgen.core.structure.Structure*)
Assigns atoms to sites for a specific structure.

This method should be implemented in the derived subclass

Parameters

- **atoms** (*list(Atom)*) – List of Atom objects to be assigned to sites.
- **struture** (*pymatgen.Structure*) – Pymatgen Structure object used to specificy the atomic coordinates.

Returns None

Notes

The atom coordinates should already be consistent with the coordinates in *structure*. Recommended usage is via the `analyse_structure()` method.

site_analysis is a Python module for analysing molecular dynamics simulations of solid-state ionic transport, by assigning mobile ions to discrete "sites" within host structures.

The code is built on top of pymatgen and takes VASP *XDATCAR* files as molecular dynamics trajectory inputs.

The code currently can use on of three schemes for assigning mobile ions to discrete sites:

1. Spherical cutoff: Atoms occupy a site if they lie within a cutoff radis from a fixed position.
2. Voronoi decomposition: Atoms are assigned to sites based on a Voronoi decomposition of the lattice into discrete volumes.
3. Polyhedral decomposition: Atoms are assigned to sites based on occupation of polyhedra defined by the instantaneous positions of lattice atoms.

CHAPTER 6

Reference

- genindex
- modindex

Python Module Index

S

Index

V

X