smact Documentation

Release 0.4.0

Author

Oct 19, 2022

Contents

View the code on Github here.

Contents:

CHAPTER 1

Introduction

smact is a collection of tools and examples for "low-fi" screening of potential semiconducting materials through the use of simple chemical rules.

smact uses a combination of heuristics and models derived from data to rapidly search large areas of chemical space. This combination of methods allows *smact* to identify new materials for applications such as photovoltaics, water splitting and thermoelectrics. Read more about *smact* in our publications:

- Computational Screening of All Stoichiometric Inorganic Materials
- Computer-aided design of metal chalcohalide semiconductors: from chemical composition to crystal structure
- Materials discovery by chemical analogy: role of oxidation states in structure prediction

This approach is heavily inspired by the work of Harrison[1] and Pamplin[2]. The work is an active project in the Walsh Materials Design Group.

SMACT is now available *via* `pip install smact`.

We are also developing a set of Jupyter notebook examples here.

[1] http://www.worldcat.org/oclc/5170450 Harrison, W. A. *Electronic structure and the properties of solids: the physics of the chemical bond* (1980)

[2] http://dx.doi.org/10.1016/0022-3697(64)90176-3 Pamplin, B. R. *J. Phys. Chem. Solids* (1964) **7** 675–684

CHAPTER 2

Getting Started

2.1 Requirements

The main language is Python 3 and has been tested using Python 3.6+. Basic requirements are Numpy and Scipy. The Atomic Simulation Environment (ASE), spglib, and pymatgen are also required for many components.

2.2 Installation

The latest stable release of SMACT can be installed via pip which will automatically setup other Python packages as required:

```
pip install smact
```

Alternatively, the latest master branch from the Git repo can be installed using:

```
pip install git+git://github.com/WMD-group/SMACT.git
```

Then ensure that the location of *smact* is on your PYTHONPATH.

For developer installation SMACT can be installed from a copy of the source repository (https://github.com/wmd-group/smact); this will be preferred if using experimental code branches.

To clone the project from Github and make a local installation:

```
git clone https://github.com/wmd-group/smact.git
cd smact
pip install --user -e .
```

With -e pip will create links to the source folder so that that changes to the code will be immediately reflected on the PATH.

CHAPTER 3

Examples

Here we will give a demonstration of how to use some of *smact*'s features. For a full set of work-through examples in Jupyter notebook form check out the examples section of our GitHub repo. For workflows that have been used in real examples and in published work, visit our separate repository.

3.1 Element and species classes

The element and species classes are at the heart of *smact*'s functionality. Elements are the elements of the periodic table. Species are elements, with some additional information; the oxidation state and the coordination environment (if known). So for example the element iron can have many oxidation states and those oxidation states can have many coordination environments.

```
import smact

iron = smact.Element('Fe')
print("The element %s has %i oxidation states. They are %s." %
(iron.symbol, len(iron.oxidation_states), iron.oxidation_states))

The element Fe has 8 oxidation states. They are [-2, -1, 1, 2, 3, 4, 5, 6].
```

When an element has an oxidation state and coordination environment then it has additional features. For example the Shannon radius[1] of the element, this is often useful for calculating radius ratio rules[2], or for training neural networks[3] .

```
iron_square_planar = smact.Species('Fe', 2, '4_n')
print('Square planar iron has a Shannon radius of %s Angstrom' % iron_square_planar.
↪shannon_radius)

Square planar iron has a Shannon radius of 0.77 Angstrom
```

[1] "Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides". Acta Crystallogr A. 32: 751–767, 1976

[2] "Crystal Structure and Chemical Constitution" Trans. Faraday Soc. 25, 253-283, 1929.

[3] "Deep neural networks for accurate predictions of crystal stability" Nat. Comms. 9, 3800, 2018.

3.2 List building

Often when using *smact* the aim will to be to search over combinations of a set of elements. This is most efficiently achieved by setting up a dictionary of the elements that you want to search over. The easiest way to achieve this in *smact* is to first create a list of the symbols of the elements that you want to include, then to build a dictionary of the corresponding element objects.

The list can be built by hand, or if you want to cover a given range there is a helper function.

```
import smact

elements = smact.ordered_elements(13, 27)
print(elements)

['Al','Si','P', 'S', 'Cl', 'Ar', 'K', 'Ca', 'Sc', 'Ti', 'V', 'Cr', 'Mn', 'Fe', 'Co']
```

For doing searches across combinations of elements it is then quickest to load the element objects into a dictionary and search by key. This avoids having to repopulate the element class at each iteration of the search.

```
element_list = smact.element_dictionary(elements)
print(element_list)

{'Al': <smact.Element at 0x10ecc5890>,
 'Ar': <smact.Element at 0x10ecc5cd0>,
 'Ca': <smact.Element at 0x10ecc5a10>,
 'Cl': <smact.Element at 0x10ecc5d90>,
 'Co': <smact.Element at 0x10ecc5f90>,
 'Cr': <smact.Element at 0x10ecc5ed0>,
 'Fe': <smact.Element at 0x10ecc5f50>,
 'K': <smact.Element at 0x10ecc5e90>,
 'Mn': <smact.Element at 0x10ecc5f10>,
 'P': <smact.Element at 0x10ecc5990>,
 'S': <smact.Element at 0x10ecc5e10>,
 'Sc': <smact.Element at 0x10ecc5150>,
 'Si': <smact.Element at 0x10e8bf190>,
 'Ti': <smact.Element at 0x10ecc5dd0>,
 'V': <smact.Element at 0x10ecc5e50>}
```

3.3 Neutral combinations

One of the most basic tests for establishing sensible combinations of elements is that they should form charge neutral combinations. This is a straightforward combinatorial problem of comparing oxidation states and allowed stoichiometries.

$\Sigma_i Q_i n_i = 0$

where i are the elements in the compound and Q are the charges. We have a special function, `smact_filter`, which does this checking for a list of elements. The `smact_filter` also ensures that all elements specified to be anions have electronegitivities greater than all elements specified to be cations.

As input `smact_filter` takes:

- `els` : a tuple of the elements to search over (required)
- `threshold`: the upper limit of the stoichiometric ratios (default = 8)

- `species_unique`: whether or not we want to consider elements in different oxidation states as unique in our results (default is False).

We can look for neutral combos.

```
import smact.screening

elements = ['Ti', 'Al', 'O']
space = smact.element_dictionary(elements)
# We just want the element items from the dictionary
eles = [e[1] for e in space.items()]
# We set a threshold for the stoichiometry of 4
allowed_combinations = smact.screening.smact_filter(eles, threshold=4)
print(allowed_combinations)
```

```
[(('Ti', 'Al', 'O'), (1, 3, 3)),
 (('Ti', 'Al', 'O'), (2, 3, 4)),
 (('Ti', 'Al', 'O'), (3, 1, 4)),
 (('Ti', 'Al', 'O'), (1, 4, 4)),
 (('Ti', 'Al', 'O'), (3, 1, 2)),
 (('Ti', 'Al', 'O'), (3, 2, 4)),
 (('Ti', 'Al', 'O'), (1, 2, 3)),
 (('Ti', 'Al', 'O'), (1, 3, 4)),
 (('Ti', 'Al', 'O'), (2, 4, 3)),
 (('Ti', 'Al', 'O'), (2, 1, 3)),
 (('Ti', 'Al', 'O'), (4, 2, 3)),
 (('Ti', 'Al', 'O'), (1, 3, 2)),
 (('Ti', 'Al', 'O'), (1, 2, 4)),
 (('Ti', 'Al', 'O'), (1, 1, 2)),
 (('Ti', 'Al', 'O'), (1, 2, 2)),
 (('Ti', 'Al', 'O'), (1, 1, 4)),
 (('Ti', 'Al', 'O'), (3, 1, 3)),
 (('Ti', 'Al', 'O'), (2, 1, 4)),
 (('Ti', 'Al', 'O'), (1, 1, 1)),
 (('Ti', 'Al', 'O'), (2, 2, 3)),
 (('Ti', 'Al', 'O'), (4, 1, 3)),
 (('Ti', 'Al', 'O'), (1, 1, 3)),
 (('Ti', 'Al', 'O'), (1, 4, 3)),
 (('Ti', 'Al', 'O'), (2, 1, 2))]
```

There is an example of how this function can be combined with multiprocessing to rapidly explore large subsets of chemical space.

3.4 Compound electronegativity

One property that is often used in high-throughput screening where band alignment is important is the compound electronegativity. Ginley and Butler showed how the simple geometric mean of the electronegitivities of a compound could be used to predict flat band potentials[4]. *smact* has a built in function to calculate this property for a given composition.

```
import smact.properties

compound_electronegs = [smact.properties.compound_electroneg(elements = a[0], stoichs␣
↪= a[1]) for \\
```

(continues on next page)

[4] "Prediction of Flatband Potentials at Semiconductor-Electrolyte Interfaces from Atomic Electronegativities" J. Electrochem. Soc. 125, 228-32, 1975.

(continued from previous page)

```
a in allowed_combinations]

print(compound_electronegs)

[4.319343517137848,
 4.729831837874991,
 4.462035251666306,
 4.337155845378665,
 5.0575817742802025,
 4.777171739263751,
 4.427325394494835,
 5.34030430325585,
 4.583732423414276,
 4.980129115226567,
 4.652147502981397,
 5.284089129411956,
 4.726884428924315,
 4.373001170931816,
 4.808336266651247,
 5.041995471272069,
 4.587722671269271,
 5.437592861777965,
 5.010966817423813,
 4.964781503487637,
 4.768922515748819,
 4.409142747625072,
 5.74200359520417,
 4.677126472294396]
```

3.5 Interfacing to machine learning

When preparing to do machine learning, we have to convert the compositions that we have into something that can be fed into an algorithm. Many of the properties provided in *smact* are suitable for this, one can take properties like electronegativity, mass, electron affinity etc etc (for the full list see *smact Python package*).

One useful representation that is often used in machine learning is the one-hot-vector formulation. A similar construction to this can be used to encode a chemical formula. A vector of length of the periodic table is set up and each element set to be a number corresponding to the stoichiometric ratio of that element in the compound. For example we could convert $Ba(OH)_2$

```
ml_vector = smact.screening.ml_rep_generator(['Ba', 'H', 'O'], stoichs=[1, 2, 2])
```

There is also an example demonstrating the conversion of charge neutral compositions produced by *smact* to a list of formulas using Pymatgen, or to a Pandas dataframe, both of which could then be used as input for a machine learning algorithm. For a full machine learning example that uses *smact*, there is a repository here which demonstrates a search for solar energy materials from the four-component (quaternary) oxide materials space.

CHAPTER 4

smact Python package

The core module of *smact* contains classes which are used as fundamental data types within the smact package, as well as several utility functions. Particular attention is drawn to *smact.element_dictionary()*, which returns a dictionary of *smact.Element* objects indexed by their chemical symbols. Generating this dictionary once and then performing lookups is generally the fastest way of accessing element data while enumerating possibilities.

Semiconducting Materials from Analogy and Chemical Theory

A collection of fast screening tools from elemental data

class smact.**Element**(*symbol*)

Bases: object

Collection of standard elemental properties for given element.

Data is drawn from "data/element.txt", part of the Open Babel package.

Atoms with a defined oxidation state draw properties from the "Species" class.

symbol

Elemental symbol used to retrieve data

Type string

name

Full name of element

Type string

number

Proton number of element

Type int

pauling_eneg

Pauling electronegativity (0.0 if unknown)

Type float

ionpot

Ionisation potential in eV (0.0 if unknown)

Type float

e_affinity
Electron affinity in eV (0.0 if unknown)

Type float

dipol
Static dipole polarizability in 1.6488e-41 C m^2 / V (0.0 if unknown)

Type float

eig
Electron eigenvalue (units unknown) N.B. For Cu, Au and Ag this defaults to d-orbital

Type float

eig_s
Eigenvalue of s-orbital

Type float

SSE
Solid State Energy

Type float

SSEPauling
SSE based on regression fit with Pauling electronegativity

Type float

oxidation_states
Default list of allowed oxidation states for use in SMACT

Type list

oxidation_states_sp
List of oxdation states recognised by the Pymatgen Structure Predictor

Type list

oxidation_states_icsd
List of oxidation states that appear in the ICSD

Type list

oxidation_states_wiki
List of oxidation states that appear wikipedia (https://en.wikipedia.org/wiki/Template:List_of_oxidation_states_of_the_elements) Data retrieved: 2022-09-22

Type list

coord_envs
The allowed coordination enviroments for the ion

Type list

covalent_radius
Covalent radius of the element

Type float

mass
Molar mass of the element

Type float

crustal_abundance
Crustal abundance in the earths crust mg/kg taken from CRC

Type float

HHI_p
Herfindahl-Hirschman Index for elemental production

Type float

HHI_r
Hirfindahl-Hirschman Index for elemental reserves

Type float

Raises

- `NameError` – Element not found in element.txt
- `Warning` – Element not found in Eigenvalues.csv

class smact.**Species**(*symbol*, *oxidation*, *coordination=4*, *radii_source='shannon'*)
Bases: *smact.Element*

Class providing data for elements in a given chemical environment

In addition to the standard properties from the periodic table (inherited from the Element class), Species objects use the oxidation state and coordination environment to provide further properties. The Species object can be created with either a default set of shannon radii (radii_source='shannon') or with a set of machine-learnt shannon radii (radii_source='extended'). The source of the machine-learnt shannon radii set is Baloch, A.A., Alqahtani, S.M., Mumtaz, F., Muqaibel, A.H., Rashkeev, S.N. and Alharbi, F.H., 2021. Extending Shannon's ionic radii database using machine learning. Physical Review Materials, 5(4), p.043804.

symbol
Elemental symbol used to retrieve data

name
Full name of element

oxidation
Oxidation state of species (signed integer)

coordination
Coordination number of species (integer)

pauling_eneg
Pauling electronegativity (0.0 if unknown)

ionpot
Ionisation potential in eV (0.0 if unknown)

e_affinity
Electron affinity in eV (0.0 if unknown)

eig
Electron eigenvalue (units unknown) N.B. For Cu, Au and Ag this defaults to d-orbital.

shannon_radius
Shannon radius of Species.

ionic_radius
Ionic radius of Species.

average_shannon_radius
An average shannon radius for the species. The average is taken over all coordination environments.

average_ionic_radius
An average ionic radius for the species. The average is taken over all coordination environments.

Raises

- `NameError` – Element not found in element.txt
- `Warning` – Element not found in Eigenvalues.csv

smact.**are_eq**(*A*, *B*, *tolerance=0.0001*)
Check two arrays for tolerance [1,2,3]==[1,2,3]; but [1,3,2]!=[1,2,3] :param A, B: 1-D list of values for approximate equality comparison :type A, B: lists :param tolerance: numerical precision for equality condition

Returns boolean

smact.**element_dictionary**(*elements=None*)
Create a dictionary of initialised smact.Element objects

Accessing an Element from a dict is significantly faster than repeadedly initialising them on-demand within nested loops.

Parameters **elements** (*iterable of strings*) – Elements to include. If None, use all elements up to 103.

Returns

Dictionary with element symbols as keys and smact.Element objects as data

Return type dict

smact.**lattices_are_same**(*lattice1*, *lattice2*, *tolerance=0.0001*)
Checks for the equivalence of two lattices

Parameters **lattice1,lattice2** – ASE crystal class

Returns boolean

smact.**neutral_ratios**(*oxidations*, *stoichs=False*, *threshold=5*)
Get a list of charge-neutral compounds

Given a list of oxidation states of arbitrary length, yield ratios in which these form a charge-neutral compound. Stoichiometries may be provided as a set of legal stoichiometries per site (e.g. a known family of compounds); otherwise all unique ratios are tried up to a threshold coefficient.

Given a list of oxidation states of arbitrary length it searches for neutral ratios in a given ratio of sites (stoichs) or up to a given threshold.

Parameters

- **oxidations** (*list of ints*) – Oxidation state of each site
- **stoichs** (*list of positive ints*) – A selection of valid stoichiometric ratios for each site
- **threshold** (*int*) – Maximum stoichiometry coefficient; if no 'stoichs' argument is provided, all combinations of integer coefficients up to this value will be tried.

Returns

exists *bool*: True ifc any ratio exists, otherwise False

allowed_ratios ***list of tuples*****:** Ratios of atoms in given oxidation states which yield a charge-neutral structure

Return type (exists, allowed_ratios) (tuple)

smact.**neutral_ratios_iter**(*oxidations*, *stoichs=False*, *threshold=5*)
Iterator for charge-neutral stoichiometries

Given a list of oxidation states of arbitrary length, yield ratios in which these form a charge-neutral compound. Stoichiometries may be provided as a set of legal stoichiometries per site (e.g. a known family of compounds); otherwise all unique ratios are tried up to a threshold coefficient.

Parameters

- **oxidations** – list of integers
- **stoichs** – stoichiometric ratios for each site (if provided)
- **threshold** – single threshold to go up to if stoichs are not provided

Yields *tuple* – ratio that gives neutrality

smact.**ordered_elements**(*x*, *y*)
Return a list of element symbols, ordered by proton number in the range x -> y :param x,y: integers

Returns Ordered list of element symbols

Return type list

4.1 Submodules

4.1.1 Structure Prediction Module

The structure prediction module implements a minimalist framework for high-throughput prediction of new compounds based on species similarity indices. The typical workflow for using this package is as follows:

1. Create a database (*StructureDB*) of *SmactStructure* s from the Materials Project.
2. Feed the database into a *StructurePredictor*, along with a list of chemical species, to predict potential likely structures by mutating structures in the database, such that they contain the given species.

4.1.1.1 Submodules

Prediction Module

Prediction algorithm implementation. Wraps the structure prediction pipeline.

Structure prediction implementation.

Todo:

- Test with a fully populated database.
- Implement n-ary substitution probabilities; at the moment, only zero- and single-species substitutions are considered.

---

class smact.structure_prediction.prediction.**StructurePredictor**(*mutator: smact.structure_prediction.mutation.Cat struct_db: smact.structure_prediction.database.Stru table: str*)

Bases: object

Provides structure prediction functionality.

Implements a statistically-based model for determining likely structures of a given composition, based on a database of known compositions and a lambda table of weights.

Based on the algorithm presented in: Hautier, G., Fischer, C., Ehrlacher, V., Jain, A., and Ceder, G. (2011) Data Mined Ionic Substitutions for the Discovery of New Compounds. Inorganic Chemistry, 50(2), 656-663. doi:10.1021/ic102031h

nary_predict_structs(*species: List[Tuple[str, int]], n_ary: Optional[int] = 2, thresh: Optional[float] = 0.001, include_same: Optional[bool] = True*) → Generator[Tuple[smact.structure_prediction.structure.SmactStructure, float, smact.structure_prediction.structure.SmactStructure], None, None]

Predicts structures for a combination of species.

Parameters

- **species** – A list of (element, charge). The constituent species of the target compound.
- **thresh** – The probability threshold, below which to discard predictions.
- **n_ary** – The number of species in a parent compound to replace.
- **include_same** – Whether to include unmodified structures from the database, i.e. structures containing all the same species.

Yields Potential structures, as tuples of (structure, probability, parent).

predict_structs(*species: List[Tuple[str, int]], thresh: Optional[float] = 0.001, include_same: Optional[bool] = True*) → Generator[Tuple[smact.structure_prediction.structure.SmactStructure, float, smact.structure_prediction.structure.SmactStructure], None, None]

Predict structures for a combination of species.

Parameters

- **species** – A list of (element, charge). The constituent species of the target compound.
- **thresh** – The probability threshold, below which to discard predictions.
- **include_same** – Whether to include unmodified structures from the database, i.e. structures containing all the same species. Defaults to True.

Yields Potential structures, as tuples of (structure, probability, parent).

Structure Database Module

Implements *StructureDB*: a minimalist SQLite database for storing *SmactStructure*-compatible POSCAR files.

Tools for database interfacing for high throughput IO.

class smact.structure_prediction.database.**StructureDB**(*db: str*)

Bases: object

SQLite Structure Database interface.

Acts as a context manager for database interfacing and wraps several useful SQLite commands within methods.

db
The database name.

conn
The database connection. Only open when used as a context manager.

cur
The database connection cursor. Only usable when class implemented as context manager.

Examples

Connecting to a database in memory:

```
>>> DB = StructureDB(':memory:')
>>> with DB as c:
...     _ = c.execute("CREATE TABLE test (id, val)")
...     c.execute("SELECT * FROM test").fetchall()
[]
>>> DB.cur.execute("SELECT * FROM test").fetchall()
Traceback (most recent call last):
    ...
sqlite3.ProgrammingError: Cannot operate on a closed database.
```

add_mp_icsd()
Add a table populated with Materials Project-hosted ICSD structures.

Note: This is very computationally expensive for large datasets and will not likely run on a laptop. If possible, download a pre-constructed database.

Parameters

- **table** (*str*) – The name of the table to add.
- **mp_data** – The Materials Project data to parse. If this is None, data will be downloaded. Downloading data needs *mp_api_key* to be set.
- **mp_api_key** (*str*) – A Materials Project API key. Only needed if *mp_data* is None.

Returns The number of structs added.

add_struct(*struct: smact.structure_prediction.structure.SmactStructure*, *table: str*)
Add a SmactStructure to a table.

Parameters

- **struct** – The *SmactStructure* to add.
- **table** – The name of the table to add the structure to.

add_structs(*structs: Sequence[smact.structure_prediction.structure.SmactStructure]*, *table: str*, *commit_after_each: Optional[bool] = False*) → int
Add several SmactStructures to a table.

Parameters

- **structs** – Iterable of *SmactStructure* s to add to table.

- **table** – The name of the table to add the structs to.
- **commit_after_each** (*bool, optional*) – Whether to commit the addition after each structure is added. This is useful when adding a large number of structures over a long timeframe, as it ensures some structures are added, even if the program terminates before completion. Defaults to False.

Returns The number of structures added.

add_table(*table: str*)
Add a table to the database.

Parameters **table** – The name of the table to add

get_structs(*composition: str*, *table: str*) → List[smact.structure_prediction.structure.SmactStructure]
Get SmactStructures for a given composition.

Parameters

- **composition** – The composition to search for. See `SmactStructure.composition()`.
- **table** – The name of the table in which to search.

Returns A list of *SmactStructure* s.

get_with_species(*species: List[Tuple[str, int]]*, *table: str*) → List[smact.structure_prediction.structure.SmactStructure]
Get SmactStructures containing given species.

Parameters

- **species** – A list of species as tuples, in (element, charge) format.
- **table** – The name of the table from which to get the species.

Returns A list of `SmactStructure` s in the table that contain the species.

smact.structure_prediction.database.**parse_mprest**()
Parse MPRester query data to generate structures.

Parameters **data** – A dictionary containing the keys 'structure' and 'material_id', with the associated values.

Returns An oxidation-state-decorated `SmactStructure`.

Mutation Module

Contains tools for determining substitution probability of different species and for mutating instances of *SmactStructure* to predict new structures.

Tools for handling ion mutation.

class smact.structure_prediction.mutation.**CationMutator**
Bases: `object`

Handles cation mutation of SmactStructures based on substitution probability.

Based on the algorithm presented in: Hautier, G., Fischer, C., Ehrlacher, V., Jain, A., and Ceder, G. (2011) Data Mined Ionic Substitutions for the Discovery of New Compounds. Inorganic Chemistry, 50(2), 656-663. doi:10.1021/ic102031h

complete_cond_probs()
Generate a DataFrame with all the conditional substitution probabilities.

complete_pair_corrs()
Generate a DataFrame with all the pair correlations.

complete_sub_probs()
Generate a DataFrame with all the substitution probabilities.

cond_sub_prob(*s1: str*, *s2: str*) → float
Calculate the probability of substitution of one species with another.

cond_sub_probs()
Calculate the probabilities of substitution of a given species.

Calculates probabilities of substitution of given species with all others in the lambda table.

static from_json(*lambda_json: Optional[str] = None, alpha: Optional[Callable[[str, str], float]] = <function CationMutator.<lambda>>*)
Create a CationMutator instance from a DataFrame.

Parameters

- **lambda_json** (*str, optional*) – JSON-style representation of the lambda table. This is a list of entries, containing pairs and their associated lambda values. Each entry is a list of [species1, species2, lambda]. If not supplied, defaults to the lambda table included with pymatgen.
- **alpha** – See __init__().

Returns A *CationMutator* instance.

get_lambda(*s1: str*, *s2: str*) → float
Get lambda values corresponding to a pair of species.

Parameters

- **s1** (*str*) – One of the species.
- **s2** (*str*) – The other species.

Returns

The lambda value, if it exists in the table. Otherwise, the alpha value for the two species.

Return type lambda (float)

get_lambdas()
Get all the lambda values associated with a species.

Parameters **species** (*str*) – The species for which to get the lambda values.

Returns A pandas Series, with index-labelled lambda entries.

pair_corr(*s1: str*, *s2: str*) → float
Determine the pair correlation of two ionic species.

same_spec_cond_probs()
Calculate the same species conditional substiution probabilities.

same_spec_probs()
Calculate the same species substiution probabilities.

sub_prob(*s1: str*, *s2: str*) → float
Calculate the probability of substitution of two species.

sub_probs()
Determine the substitution probabilities of a species with others.

Determines the probability of substitution of the species with every species in the lambda table.

unary_substitute(*structure: smact.structure_prediction.structure.SmactStructure*, *thresh: Optional[float] = 1e-05*) → Generator[Tuple[smact.structure_prediction.structure.SmactStructure, float], None, None]

Find all structures with 1 substitution with probability above a threshold.

Parameters

- **structure** – A `SmactStructure` instance from which to generate compounds.
- **thresh** (*float*) – The probability threshold; discard all substitutions that have a probability to generate a naturally-occuring compound less than this.

Yields Tuples of (`SmactStructure`, probability).

Structure Module

Contains a minimalist *SmactStructure* class for simple crystal structure and chemical composition representation and manipulation.

Minimalist structure representation for comprehensible manipulation.

class smact.structure_prediction.structure.**SmactStructure**

Bases: `object`

SMACT implementation inspired by pymatgen Structure class.

Handles basic structural and compositional information for a compound. Includes a lossless POSCAR-style specification for storing structures, allowing structures to be stored in files or databases, or to be pulled from the Materials Project.

species

A list of tuples describing the composition of the structure, stored as (element, oxidation, stoichiometry). The list is sorted alphabetically based on element symbol, and identical elements are sorted with highest charge first.

lattice_mat

A numpy 3x3 array containing the lattice vectors.

sites

A dictionary of {species: coords}, where species is a string representation of the species and coords is a list of position vectors, given as lists of length 3. For example:

```
>>> s = SmactStructure.from_file('tests/files/NaCl.txt')
>>> s.sites
{'Cl1-': [[2.323624165, 1.643050405, 4.02463512]], 'Na1+': [[0.0, 0.0, 0.0]]}
```

lattice_param

The lattice parameter.

as_poscar() → str

Represent the structure as a POSCAR file compatible with VASP5.

The POSCAR format adopted is as follows:

The first line contains the species' names separated by a whitespace. The second through fourth line, inclusive, contain the lattice matrix: each line contains a lattice vector, with elements separated by a whitespace. The fifth line contains the elements' names separated by a whitespace. If more than one oxidation state exists for an element, the element appears multiple times; once for each oxidation state. The sixth line is the string 'Cartesian'. The seventh line onwards are the Cartesian coordinates of each

site, separated by a whitespace. In addition, at the end of each line is the species' name, separated by a whitespace.

For examples of this format, see the text files under tests/files.

Returns POSCAR-style representation of the structure.

Return type str

composition() → str
Generate a key that describes the composition.

Key format is '{element}_{stoichiometry}_{charge}{sign}' with no delimiter, *sans brackets*. Species are ordered as stored within the structure, see *SmactStructure*.

Returns Key describing constituent species.

Examples

```
>>> s = SmactStructure.from_file('tests/files/CaTiO3.txt')
>>> print(s.composition())
Ca_1_2+O_3_2-Ti_1_4+
```

static from_file(*fname: str*)
Create SmactStructure from a POSCAR file.

Parameters **fname** – The name of the POSCAR file. See *as_poscar()* for format specification.

Returns *SmactStructure*

static from_mp(*species: List[Union[Tuple[str, int, int], Tuple[smact.Species, int]]], api_key: str*)
Create a SmactStructure using the first Materials Project entry for a composition.

Parameters

- **species** – See __init__().
- **api_key** – A www.materialsproject.org API key.

Returns *SmactStructure*

static from_poscar(*poscar: str*)
Create SmactStructure from a POSCAR string.

Parameters **poscar** – A SMACT-formatted POSCAR string. See *as_poscar()* for format specification.

Returns *SmactStructure*

static from_py_struct()
Create a SmactStructure from a pymatgen Structure object.

Parameters **structure** – A pymatgen Structure.

Returns *SmactStructure*

get_spec_strs() → List[str]
Get string representations of the constituent species.

Returns A list of strings, formatted as '{element}{charge}{sign}'.

Examples

```
>>> s = SmactStructure.from_file('tests/files/CaTiO3.txt')
>>> s.get_spec_strs()
['Ca2+', 'O2-', 'Ti4+']
```

has_species (*species: Tuple[str, int]*) → bool
Determine whether a given species is in the structure.

Substitution Probability Models

Minimal API for developing substitution likelihood probability models for ion mutation.

Probability models for species substitution.

Implements base class *SubstitutionModel*, which can be extended to allow for development of new lambda tables. An example of such an extension, *RadiusModel*, is also implemented.

Todo:

- Allow for parallelism in lambda table calculations by implementing a *sub_probs* abstractmethod that *SubstitutionModel.gen_lambda()* uses, if available.

---

class smact.structure_prediction.probability_models.**RadiusModel**
Bases: *smact.structure_prediction.probability_models.SubstitutionModel*

Substitution probability model based on Shannon radii.

sub_prob (*s1*, *s2*)
Calculate the probability of substituting species s1 for s2.

Based on the difference in Shannon radii, the probability is assumed to be:

$$p = 1 - k\Delta r^2.$$

Parameters

- **s1** – The species being substituted.
- **s2** – The species substituting.

Returns The probability of substitution.

class smact.structure_prediction.probability_models.**SubstitutionModel**
Bases: abc.ABC

Abstract base class for substitution models.

gen_lambda()
Generate a lambda table for a list of species.

Parameters **species** – A list of species strings.

Returns A pivot table-style DataFrame containing lambda values for every possible species pair.

sub_prob (*s1: str*, *s2: str*) → float
Calculate the probability of substituting species s1 for s2.

Parameters

- **s1** – The species being substituted.

- **s2** – The species substituting.

Returns The probability of substitution.

Structure Prediction Utilities Module

Miscellaneous utilities for data manipulation.

Miscellaneous tools for data parsing.

smact.structure_prediction.utilities.**get_sign**(*charge: int*) → str
Get string representation of a number's sign.

Parameters **charge** (*int*) – The number whose sign to derive.

Returns either '+', '-', or '' for neutral.

Return type sign (str)

smact.structure_prediction.utilities.**parse_spec**(*species: str*) → Tuple[str, int]
Parse a species string into its element and charge.

Parameters **species** (*str*) – String representation of a species in the format {element}{absolute_charge}{sign}.

Returns A tuple of (element, signed_charge).

Examples

```
>>> parse_spec("Fe2+")
('Fe', 2)
>>> parse_spec("O2-")
('O', -2)
```

smact.structure_prediction.utilities.**unparse_spec**(*species: Tuple[str, int]*) → str
Unparse a species into a string representation.

The analogue of *parse_spec()*.

Parameters **tuple of** (*A*) –

Returns String of {element}{absolute_charge}{sign}.

Examples

```
>>> unparse_spec(("Fe", 2))
'Fe2+'
>>> unparse_spec(("O", -2))
'O2-'
```

4.1.2 Dopant Prediction Module

Semiconductor dopant screening

The dopant prediction module helps identify possible n-type p-type dopants.

1. *doper*, A class to identify possible n-type p-type dopants + give suggestions of original species given as a dictionary with "n_type_cation", "p_type_cation", "n_type_anion", "p_type_anion", and it calculates the corresponding substitution probabilities.

4.1.2.1 Submodules

smact.dopant_prediction.doper module

A class to create ntype ptype possiblie dopants for input species

class smact.dopant_prediction.doper.**Doper**(*original_species: Tuple[str], num_dopants=5*)
Bases: object

A class to search for n and p type dopants Methods: get_dopants, plot_dopants

get_dopants() → dict
Note currently limited to binary compounds

Parameters

- **get_dopants**(*ex*) –
- **original_species**(*tuple(str)* = *('Cd2+', 'O2-')* –
- **num_dopants**(*int*) –
- **match_oxi_sign**(*bool*) –

Returns Dopant suggestions, given as a dictionary with keys "n_type_cation", "p_type_cation", "n_type_anion", "p_type_anion".

Return type (dict)

plot_dopants()
Uses pymatgen plotting utilities to plot the results of the dopant search

4.1.3 smact.properties module

A collection of tools for estimating useful properties.

The "electronegativity of a compound" computed with compound_electroneg() is the rescaled geometric mean of electronegativity used in Nethercot's recipe for estimating the photoelectric threshold:[1]

$$\Phi^{AB} = 2.86(\chi_A \chi_B)^{1/2} + E_g/2.$$

In other words, the computed group $2.86(\chi_A \chi_B)^{1/2}$ is the mid-gap energy and the VBM/CBM positions can be estimated by subtracting/adding half of the band gap E_g. This is an extension Mulliken's electronegativity scale in which $\chi_A = (I_A + E_A)/2$ (where I and E are respectively the ionisation potential and electron affinity.)[2]

smact.properties.**band_gap_Harrison**(*anion*, *cation*, *verbose=False*, *distance=None*)
Estimates the band gap from elemental data.

The band gap is estimated using the principles outlined in Harrison's 1980 work "Electronic Structure and the Properties of Solids: The Physics of the Chemical Bond".

Parameters

- **Anion**(*str*) – Element symbol of the dominant anion in the system

[1] Nethercot, A. H. (1974). *Phys. Rev. Lett.*, **33**, 1088–1091. http://dx.doi.org/10.1103/PhysRevLett.33.1088
[2] Mulliken, R. S. (1934). *J. Chem. Phys.*, **2**, 782. http://dx.doi.org/10.1063/1.1749394

- **Cation** (*str*) – Element symbol of the the dominant cation in the system
- **Distance** (*float or str*) – Nuclear separation between anion and cation i.e. sum of ionic radii
- **verbose** (*bool*) – An optional True/False flag. If True, additional
- **is printed to the standard output. [Defult** (*information*) – False]

Returns : Band_gap (float): Band gap in eV

smact.properties.**compound_electroneg**(*verbose=False*, *elements=None*, *stoichs=None*, *source='Mulliken'*)

Estimate electronegativity of compound from elemental data.

Uses Mulliken electronegativity by default, which uses elemental ionisation potentials and electron affinities. Alternatively, can use Pauling electronegativity, re-scaled by factor 2.86 to achieve same scale as Mulliken method (Nethercot, 1974) DOI:10.1103/PhysRevLett.33.1088 .

Geometric mean is used (n-th root of product of components), e.g.:

X_Cu2S = (X_Cu * X_Cu * C_S)^(1/3)

Parameters

- **elements** (*list*) – Elements given as standard elemental symbols.
- **stoichs** (*list*) – Stoichiometries, given as integers or floats.
- **verbose** (*bool*) – An optional True/False flag. If True, additional information is printed to the standard output. [Default: False]
- **source** – String 'Mulliken' or 'Pauling'; type of Electronegativity to use. Note that in SMACT, Pauling electronegativities are rescaled to a Mulliken-like scale.

Returns Estimated electronegativity (no units).

Return type Electronegativity (float)

smact.properties.**eneg_mulliken**(*element*)

Get Mulliken electronegativity from the IE and EA.

Parameters **symbol** (smact.Element *or str*) – Element object or symbol

Returns Mulliken electronegativity

Return type mulliken (float)

4.1.4 smact.screening module

smact.screening.**eneg_states_test**(*ox_states*, *enegs*)

Internal function for checking electronegativity criterion

This implementation is fast as it 'short-circuits' as soon as it finds an invalid combination. However it may be that in some cases redundant comparisons are made. Performance is very close between this method and eneg_states_test_alternate.

Parameters

- **ox_states** (*list*) – oxidation states corresponding to species in compound
- **enegs** (*list*) – Electronegativities corresponding to species in compound

Returns

True if cations have higher electronegativity than anions, otherwise False

Return type bool

smact.screening.**eneg_states_test_alternate**(*ox_states*, *enegs*)
Internal function for checking electronegativity criterion

This implementation appears to be slightly slower than eneg_states_test, but further testing is needed.

Parameters

- **ox_states** (*list*) – oxidation states corresponding to species in compound
- **enegs** (*list*) – Electronegativities corresponding to species in compound

Returns

True if cations have higher electronegativity than anions, otherwise False

Return type bool

smact.screening.**eneg_states_test_threshold**(*ox_states*, *enegs*, *threshold=0*)
Internal function for checking electronegativity criterion

This implementation is fast as it 'short-circuits' as soon as it finds an invalid combination. However it may be that in some cases redundant comparisons are made. Performance is very close between this method and eneg_states_test_alternate.

A 'threshold' option is added so that this constraint may be relaxed somewhat.

Parameters

- **ox_states** (*list*) – oxidation states corresponding to species in compound
- **enegs** (*list*) – Electronegativities corresponding to species in compound
- **threshold** (*Option(float)*) – a tolerance for the allowed deviation from the Pauling criterion

Returns

True if cations have higher electronegativity than anions, otherwise False

Return type bool

smact.screening.**ml_rep_generator**(*composition*, *stoichs=None*)
Function to take a composition of Elements and return a list of values between 0 and 1 that describes the composition, useful for machine learning.

The list is of length 103 as there are 103 elements considered in total in SMACT.

e.g. Li2O –> [0, 0, 2/3, 0, 0, 0, 0, 1/3, 0 ....]

Inspired by the representation used by Legrain et al. DOI: 10.1021/acs.chemmater.7b00789

Parameters

- **composition** (*list*) – Element objects in composition OR symbols of elements in composition
- **stoichs** (*list*) – Corresponding stoichiometries in the composition

Returns

List of floats representing the composition that sum to one

Return type norm (list)

smact.screening.**pauling_test**(*oxidation_states*, *electronegativities*, *symbols=[]*, *repeat_anions=True*, *repeat_cations=True*, *threshold=0.0*)

Check if a combination of ions makes chemical sense, (i.e. positive ions should be of lower electronegativity).

Parameters

- **ox** (*list*) – oxidation states of elements in the compound
- **paul** (*list*) – the corresponding Pauling electronegativities of the elements in the compound
- **symbols** (*list*) – chemical symbols of each site
- **threshold** (*float*) – a tolerance for the allowed deviation from the Pauling criterion
- **repeat_anions** – boolean, allow an anion to repeat in different oxidation states in the same compound
- **repeat_cations** – as above, but for cations

Returns True if positive ions have lower electronegativity than negative ions

Return type bool

smact.screening.**pauling_test_old**(*ox*, *paul*, *symbols*, *repeat_anions=True*, *repeat_cations=True*, *threshold=0.0*)

Check if a combination of ions makes chemical sense, (i.e. positive ions should be of lower Pauling electronegativity). This function should give the same results as pauling_test but is not optimised for speed.

Parameters

- **ox** (*list*) – oxidation states of the compound
- **paul** (*list*) – the corresponding Pauling electronegativities of the elements in the compound
- **symbols** (*list*) – chemical symbols of each site.
- **threshold** (*float*) – a tolerance for the allowed deviation from the Pauling criterion
- **repeat_anions** – boolean, allow an anion to repeat in different oxidation states in the same compound.
- **repeat_cations** – as above, but for cations.

Returns True if positive ions have lower electronegativity than negative ions

Return type (bool)

smact.screening.**smact_filter**(*els*, *threshold=8*, *species_unique=True*, *oxidation_states_set='default'*)

Function that applies the charge neutrality and electronegativity tests in one go for simple application in external scripts that wish to apply the general 'smact test'.

Parameters

- **els** (*tuple/list*) – A list of smact.Element objects
- **threshold** (*int*) – Threshold for stoichiometry limit, default = 8
- **species_unique** (*bool*) – Whether or not to consider elements in different oxidation states as unique in the results.

- **oxidation_states_set** (*string*) – A string to choose which set of oxidation states should be chosen. Options are 'default', 'icsd', 'pymatgen' and 'wiki' for the default, icsd, pymatgen structure predictor and Wikipedia (https://en.wikipedia.org/wiki/Template:List_of_oxidation_states_of_the_elements) oxidation states.

Returns Allowed compositions for that chemical system in the form [(elements), (oxidation states), (ratios)] if species_unique=True or in the form [(elements), (ratios)] if species_unique=False.

Return type allowed_comps (list)

4.1.5 smact.oxidation_states module

smact.oxidation_states: Module for predicting the likelihood of species coexisting in a compound based on statistical analysis of oxidation states. It is possible to use the values obtained in the publication Materials Discovery by Chemical Analogy: Role of Oxidation States in Structure Prediction - DOI: 10.1039/C8FD00032H.

class smact.oxidation_states.**Oxidation_state_probability_finder**(*probability_table=None*)
Bases: object

Uses the model developed in the Faraday Discussions Paper (DOI:10.1039/C8FD00032H) to compute the likelihood of metal species existing in solids in the presence of certain anions.

compound_probability(*structure*, *ignore_stoichiometry=True*)
calculate overall probability for structure or composition.

Parameters

- **structure** (*pymatgen.Structure*) – Compound for which the probability score will be generated. Can also be a list of pymatgen or SMACT Species.
- **ignore_stoichiometry** (*bool*) – Whether to weight probabilities by stoichiometry. Defaults to false as decribed in the original paper.

Returns Compound probability

Return type compound_prob (float)

get_included_species()
Returns a list of species for which there exists data in the probability table used.

pair_probability(*species1*, *species2*)
Get the anion-cation oxidation state probability for a provided pair of smact Species. i.e. $P_{SA} = \frac{N_{SX}}{N_{MX}}$ in the original paper (DOI:10.1039/C8FD00032H).

Parameters

- **species1** (smact.Species) – Cation or anion species
- **species2** (smact.Species) – Cation or anion species

Returns Species-anion probability

Return type prob (float)

4.1.6 smact.builder module

A collection of functions for building certain lattice types. Currently there are examples here for the Perovskite and Wurzite lattice types, which rely on the Atomic Simulation Environment (ASE) spacegroup.crystal() function.

smact.builder.**cubic_perovskite**(*species, cell_par=[6, 6, 6, 90, 90, 90], repetitions=[1, 1, 1]*)
Build a perovskite cell using the crystal function in ASE.

Parameters

- **species** (*str*) – Element symbols
- **cell_par** (*list*) – Six floats/ints specifying 3 unit cell lengths and 3 unit cell angles.
- **repetitions** (*list*) – Three floats specifying the expansion of the cell in x,y,z directions.

Returns SMACT Lattice object of the unit cell, ASE crystal system of the unit cell.

smact.builder.**wurtzite**(*species, cell_par=[2, 2, 6, 90, 90, 120], repetitions=[1, 1, 1]*)
Build a wurzite cell using the crystal function in ASE.

Parameters

- **species** (*str*) – Element symbols
- **cell_par** (*list*) – Six floats/ints specifying 3 unit cell lengths and 3 unit cell angles.
- **repetitions** (*list*) – Three floats specifying the expansion of the cell in x,y,z directions.

Returns SMACT Lattice object of the unit cell, ASE crystal system of the unit cell.

4.1.7 smact.distorter module

smact.distorter: Module for generating symmetry-unique substitutions on a given sub-lattice.

As input it takes the ASE crystal object (as built by smact.builder) and the sub-lattice on which substitutions are to be made. There is an example of how to use the code in Example_distort.py

TODO: Add a functionality to check two Atoms objects against one another for equivalence.

smact.distorter.**build_sub_lattice**(*lattice*, *symbol*)
Generate a sub-lattice of the lattice based on equivalent atomic species.

Parameters

- **lattice** (*ASE crystal class*) – Input lattice
- **symbol** (*string*) – Symbol of species identifying sub-lattice

Returns sub_lattice: Cartesian coordinates of the sub-lattice of symbol

Return type list of lists

smact.distorter.**get_inequivalent_sites**(*sub_lattice*, *lattice*)
Given a sub lattice, returns symmetry unique sites for substitutions.

Parameters

- **sub_lattice** (*list of lists*) – array containing Cartesian coordinates of the sub-lattice of interest
- **lattice** (*ASE crystal*) – the total lattice

Returns List of sites

smact.distorter.**get_sg**(*lattice*)
Get the space-group of the system.

Parameters **lattice** – the ASE crystal class

Returns integer number of the spacegroup

Return type sg (int)

smact.distorter.**make_substitution**(*lattice*, *site*, *new_species*)
Change atomic species on lattice site to new_species.

Parameters

- **lattice** (*ASE crystal*) – Input lattice
- **site** (*list*) – Cartesian coordinates of the substitution site
- **new_species** (*str*) – New species

Returns lattice

4.1.8 smact.lattice module

class smact.lattice.**Lattice**(*sites*, *space_group=1*, *strukturbericht=False*)
Bases: object

A unique set of Sites.

Lattice objects define a general crystal structure, with a space group and a collection of Site objects. These Site objects have their own fractional coordinates and a list of possible oxidation states (see the Site class).

Specific crystal structures with elements assigned to sites are "materials" and use the Atoms class from the Atomic Simulation Environment.

basis_sites
A list of Site objects [SiteA, SiteB, SiteC, . . .]

comprising the basis sites in Cartesian coordinates

space_group
Integer space group number according to the

International Tables for Crystallography.

structurbericht

Structurbericht identity, if applicable

Type e.g. 'B1'

lattice_vector_calc()

class smact.lattice.**Site**(*position, oxidation_states=[0]*)
Bases: object

A single lattice site with a list of possible oxidation states.

The Site object is primarily used within Lattice objects.

position
A list of fractional coordinates [x,y,z]

oxidation_states
A list of possible oxidation states e.g. [-1,0,1]

4.1.9 smact.lattice_parameters module

This module can be used to calculate roughly the lattice parameters of a lattice type, based on the radii of the species on each site.

smact.lattice_parameters.**b10**(*shannon_radius*)
The lattice parameters of Litharge

Parameters **shannon_radius** (*list*) – The radii of the a,b ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**b2**(*shannon_radius*)
The lattice parameters of b2.

Parameters **shannon_radius** (*list*) – The radii of the a,b ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**bcc**(*covalent_radius*)
The lattice parameters of the A2.

Parameters **shannon_radius** (*list*) – The radii of the a ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**bct**(*covalent_radius*)
The lattice parameters of the bct.

Parameters **shannon_radius** (*list*) – The radii of the a ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**cubic_perovskite**(*shannon_radius*)
The lattice parameters of the cubic perovskite structure.

Parameters **shannon_radius** (*list*) – The radii of the a,b,c ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**diamond**(*covalent_radius*)
The lattice parameters of the diamond.

Parameters **shannon_radius** (*list*) – The radii of the a ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**fcc**(*covalent_radius*)
The lattice parameters of the A1.

Args: shannon_radius (list) : The radii of the a ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**hcp**(*covalent_radius*)
The lattice parameters of the hcp.

Parameters **shannon_radius** (*list*) – The radii of the a ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**rocksalt**(*shannon_radius*)
The lattice parameters of rocksalt.

Parameters **shannon_radius** (*list*) – The radii of the a,b ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**stuffed_wurtzite**(*shannon_radii*)
The stuffed wurtzite structure (e.g. LiGaGe) space group P63/mc.

Parameters **shannon_radius** (*list*) – The radii of the a,b,c ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**wurtzite**(*shannon_radius*)
The lattice parameters of the wurtzite structure.

Parameters **shannon_radius** (*list*) – The radii of the a,b ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

smact.lattice_parameters.**zincblende**(*shannon_radius*)

The lattice parameters of Zinc Blende. Args: shannon_radius (list) : The radii of the a,b ions

Returns float values of lattics constants and angles (a, b, c, alpha, beta, gamma)

Return type (tuple)

4.1.10 smact.data_loader module

Provide data from text files while transparently caching for efficiency.

This module handles the loading of external data used to initialise the core smact.Element and smact.Species classes. It implements a transparent data-caching system to avoid a large amount of I/O when naively constructing several of these objects. It also implements a switchable system to print verbose warning messages about possible missing data (mainly for debugging purposes). In general these fuctions are used in the background and it is not necessary to use them directly.

smact.data_loader.**float_or_None**(*x*)
Cast a string to a float or to a None.

smact.data_loader.**lookup_element_data**(*symbol*, *copy=True*)
Retrieve tabulated data for an element.

The table "data/element_data.txt" contains a collection of relevant atomic data. If a cache exists in the form of the module-level variable _element_data, this is returned. Otherwise, a dictionary is constructed from the data table and cached before returning it.

Parameters

- **symbol** (*str*) – Atomic symbol for lookup
- **copy** (*Optional(bool)*) – if True (default), return a copy of the data dictionary, rather than a reference to the cached object – only used copy=False in performance-sensitive code and where you are certain the dictionary will not be modified!

Returns (dict) [Dictionary of data for given element, keyed by] column headings from data/element_data.txt.

smact.data_loader.**lookup_element_hhis**(*symbol*)

Retrieve the HHI_R and HHI_p scores for an element.

Parameters **symbol** – the atomic symbol of the element to look up.

Returns

(HHI_p, HHI_R)

Return None if values for the elements were not found in the external data.

Return type tuple

smact.data_loader.**lookup_element_oxidation_states**(*symbol*, *copy=True*)

Retrieve a list of known oxidation states for an element. The oxidation states list used is the SMACT default and most exhaustive list.

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the oxidation-state list, rather than a reference to the cached data – only use copy=False in performance-sensitive code and where the list will not be modified!

Returns

List of known oxidation states for the element.

Returns None if oxidation states for the Element were not found in the external data.

Return type list

smact.data_loader.**lookup_element_oxidation_states_icsd**(*symbol*, *copy=True*)

Retrieve a list of known oxidation states for an element. The oxidation states list used contains only those found in the ICSD (and judged to be non-spurious).

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the oxidation-state list, rather than a reference to the cached data – only use copy=False in performance-sensitive code and where the list will not be modified!

Returns

List of known oxidation states for the element.

Return None if oxidation states for the Element were not found in the external data.

Return type list

smact.data_loader.**lookup_element_oxidation_states_sp**(*symbol*, *copy=True*)

Retrieve a list of known oxidation states for an element. The oxidation states list used contains only those that are in the Pymatgen default lambda table for structure prediction.

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the oxidation-state list, rather than a reference to the cached data – only use copy=False in performance-sensitive code and where the list will not be modified!

Returns

List of known oxidation states for the element.

Return None if oxidation states for the Element were not found in the external data.

Return type list

smact.data_loader.**lookup_element_oxidation_states_wiki**(*symbol*, *copy=True*)

Retrieve a list of known oxidation states for an element. The oxidation states list used contains only those that are on Wikipedia (https://en.wikipedia.org/wiki/Template:List_of_oxidation_states_of_the_elements).

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the oxidation-state list, rather than a reference to the cached data – only use copy=False in performance-sensitive code and where the list will not be modified!

Returns

List of known oxidation states for the element.

Return None if oxidation states for the Element were not found in the external data.

Return type list

smact.data_loader.**lookup_element_shannon_radius_data**(*symbol*, *copy=True*)

Retrieve Shannon radii for known states of an element.

Retrieve Shannon radii for known oxidation states and coordination environments of an element.

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the data
- **rather than a reference to the cached object --** (*dictionary,*) –
- **use copy=False in performance-sensitive code and where** (*only*) –
- **are certain the dictionary will not be modified!** (*you*) –

Returns

Shannon radii datasets.

Returns None if the element was not found among the external data.

Shannon radii datasets are dictionaries with the keys:

charge *int* charge

coordination *int* coordination

crystal_radius *float*

ionic_radius *float*

comment *str*

Return type list

smact.data_loader.**lookup_element_shannon_radius_data_extendedML**(*symbol*, *copy=True*)

Retrieve the machine learned extended Shannon radii for known states of an element.

Retrieve Shannon radii for known oxidation states and coordination environments of an element.

Source of extended radii is: Baloch, A.A., Alqahtani, S.M., Mumtaz, F., Muqaibel, A.H., Rashkeev, S.N. and Alharbi, F.H., 2021. Extending Shannon's Ionic Radii Database Using Machine Learning. arXiv preprint arXiv:2101.00269.

Parameters

- **symbol** (*str*) – the atomic symbol of the element to look up.
- **copy** (*Optional(bool)*) – if True (default), return a copy of the data
- **rather than a reference to the cached object --** (*dictionary,*) –
- **use copy=False in performance-sensitive code and where** (*only*) –
- **are certain the dictionary will not be modified!** (*you*) –

Returns

Extended Shannon radii datasets.

Returns None if the element was not found among the external data.

Shannon radii datasets are dictionaries with the keys:

charge *int* charge

coordination *int* coordination

ionic_radius *float*

comment *str*

Return type list

smact.data_loader.**lookup_element_sse2015_data**(*symbol*, *copy=True*)

Retrieve SSE (2015) data for element in oxidation state.

Retrieve the solid-state energy (SSE2015) data for an element in an oxidation state. Taken from J. Solid State Chem., 2015, 231, pp138-144, DOI: 10.1016/j.jssc.2015.07.037.

Parameters

- **symbol** – the atomic symbol of the element to look up.
- **copy** – if True (default), return a copy of the data dictionary,
- **than a reference to a cached object -- only use** (*rather*) –
- **in performance-sensitive code and where you are** (*copy=False*) –
- **the dictionary will not be modified!** (*certain*) –

Returns

SSE datasets for the element, or None if the element was not found among the external data.

SSE datasets are dictionaries with the keys:

OxidationState *int*

SolidStateEnergy2015 *float* SSE2015

Return type list

smact.data_loader.**lookup_element_sse_data**(*symbol*)

Retrieve the solid-state energy (SSE) data for an element.

Taken from J. Am. Chem. Soc., 2011, 133 (42), pp 16852-16960, DOI: 10.1021/ja204670s

Parameters **symbol** – the atomic symbol of the element to look up.

Returns

SSE datasets for the element, or None if the element was not found among the external data.

SSE datasets are dictionaries with the keys:

AtomicNumber *int*

SolidStateEnergy *float* SSE

IonisationPotential *float*

ElectronAffinity *float*

MullikenElectronegativity *str*

SolidStateRenormalisationEnergy *float*

Return type list

smact.data_loader.**lookup_element_sse_pauling_data**(*symbol*)

Retrieve Pauling SSE data

Retrieve the solid-state energy (SSEPauling) data for an element from the regression fit when SSE2015 is plotted against Pauling electronegativity. Taken from J. Solid State Chem., 2015, 231, pp138-144, DOI: 10.1016/j.jssc.2015.07.037

Args: symbol (str) : the atomic symbol of the element to look up.

Returns: A dictionary containing the SSE2015 dataset for the element, or None if the element was not found among the external data.

smact.data_loader.**set_warnings**(*enable=True*)

Set verbose warning messages on and off.

In order to see any of the warnings, this function needs to be called _before_ the first call to the smact.Element() constructor.

Args: enable (bool) : print verbose warning messages.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

O

P

R

S

U

W

Z