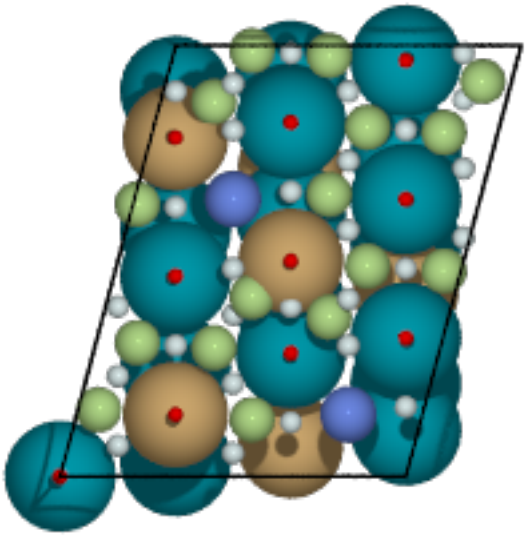
CatKit

Release 0.5.0

Aug 29, 2018

Contents

Welcome to CatKit! A staging ground for computational tools which are generally useful for catalysis research. The goal of the project is to provide a communal location for those interested in hosting such tools under a common banner. In doing so, we hope to provide the infrastructure to produce more advanced functionality based on modular components of individual contributors.



CHAPTER 1

Contents

1.1 Modules

1.1.1 Subpackages

catkit.gen package

Subpackages

catkit.gen.analysis package

Submodules

catkit.gen.analysis.classifier module

class catkit.gen.analysis.classifier.**Classifier**(*atoms*)

Class for classification of various aspects of an an atomic unit cell.

Currently, a tool for classification of adsorbates on surface environments and the active sites they rest on.

id_adsorbate_atoms(*classifier='trivial'*, *tag=False*)

Identify adsorbed atoms in a given atoms object.

Parameters

- **classifier** (*str*) – Classification technique to identify adsorbate atoms.

  'trivial': Adsorbate atoms assumed to have atomic number != 13 or < 21.

- **tag** (*bool*) – Return adsorbate atoms with tags of -2.

Returns **ads_atoms** – Index of adsorbate atoms found.

Return type ndarray (n,)

id_adsorbates(*classifier='radial'*, *return_atoms=False*)
Return a list of Gratoms objects for each adsorbate classified on a surface. Requires classification of adsorbate atoms.

Parameters

- **classifier** (*str*) – Classification technique to identify individual adsorbates.

  'radial': Use standard cutoff distances to identify neighboring atoms.

- **return_atoms** (*bool*) – Return Gratoms objects instead of adsorbate indices.

Returns **adsorbates** – Adsorbate indices of adsorbates in unit cell.

Return type list (n,)

id_slab_atoms(*classifier='trivial'*, *tag=False*, *rtol=0.001*)
Return the indices of the slab atoms using select characterization techniques.

Parameters

- **classifier** (*str*) – Classification technique to identify slab atoms.

  'trivial': Slab atoms assumed to have atomic number == 13 or >= 21.

- **tag** (*bool*) – Return adsorbate atoms with tags of 2.
- **rtol** (*float*) – Relative cutoff distance for tagging layers.

Returns **slab_atoms** – Index of slab atoms found.

Return type ndarray (n,)

id_surface_atoms(*classifier='voronoi_sweep'*)
Identify surface atoms of an atoms object. This will require that adsorbate atoms have already been identified.

Parameters **classifier** (*str*) – Classification technique to identify surface atoms.

'voronoi_sweep': Create a sweep of proxy atoms above surface. Surface atoms are those which are most frequent neighbors of the sweep.

Returns **surface_atoms** – Index of the surface atoms in the object.

Return type ndarray (n,)

catkit.gen.analysis.matching module

catkit.gen.analysis.matching.**reactant_indices**(*R1*, *R2*, *P*, *broken_bond*)

Match the indices of a pair of reactants from a product and broken bond.

Parameters

- **R1** (*networkx MultiGraph*) – Graph representing reactant 1
- **R2** (*networkx MultiGraph*) – Graph representing reactant 2
- **P** (*networkx MultiGraph*) – Graph representing the product
- **broken_bond** (*list (2,)*) – Indices representing the edge of the product to be removed.

Returns **pindex** – Indices of the product graph sorted by the order of the reactants indices.

Return type ndarrays (n,)

catkit.gen.analysis.matching.**slab_indices**(*slab0*, *slab1*, *mask=None*)
Match the indices of similar atoms between two slabs.

catkit.gen.analysis.reconfiguration module

catkit.gen.analysis.reconfiguration.**id_reconstruction**(*images*, *save=False*)
Identify a reconstruction event by analyzing changes in the forces.

Parameters

- **images** (*list of ASE atoms objects*) – Relaxation trajectory.
- **show** (*bool*) – Create a figure to display the events located.

Returns **predicted_events** – index of images predicted before the event occurs.

Return type list of int

Module contents

class catkit.gen.analysis.**Classifier**(*atoms*)
Class for classification of various aspects of an an atomic unit cell.

Currently, a tool for classification of adsorbates on surface environments and the active sites they rest on.

id_adsorbate_atoms(*classifier='trivial'*, *tag=False*)
Identify adsorbed atoms in a given atoms object.

Parameters

- **classifier** (*str*) – Classification technique to identify adsorbate atoms.

  'trivial': Adsorbate atoms assumed to have atomic number != 13 or < 21.

- **tag** (*bool*) – Return adsorbate atoms with tags of -2.

Returns **ads_atoms** – Index of adsorbate atoms found.

Return type ndarray (n,)

id_adsorbates(*classifier='radial'*, *return_atoms=False*)
Return a list of Gratoms objects for each adsorbate classified on a surface. Requires classification of adsorbate atoms.

Parameters

- **classifier** (*str*) – Classification technique to identify individual adsorbates.

  'radial': Use standard cutoff distances to identify neighboring atoms.

- **return_atoms** (*bool*) – Return Gratoms objects instead of adsorbate indices.

Returns **adsorbates** – Adsorbate indices of adsorbates in unit cell.

Return type list (n,)

id_slab_atoms(*classifier='trivial'*, *tag=False*, *rtol=0.001*)
Return the indices of the slab atoms using select characterization techniques.

Parameters

- **classifier** (*str*) – Classification technique to identify slab atoms.

  'trivial': Slab atoms assumed to have atomic number == 13 or >= 21.

- **tag** (*bool*) – Return adsorbate atoms with tags of 2.
- **rtol** (*float*) – Relative cutoff distance for tagging layers.

Returns **slab_atoms** – Index of slab atoms found.

Return type ndarray (n,)

id_surface_atoms(*classifier='voronoi_sweep'*)

Identify surface atoms of an atoms object. This will require that adsorbate atoms have already been identified.

Parameters **classifier** (*str*) – Classification technique to identify surface atoms.

'voronoi_sweep': Create a sweep of proxy atoms above surface. Surface atoms are those which are most frequent neighbors of the sweep.

Returns **surface_atoms** – Index of the surface atoms in the object.

Return type ndarray (n,)

Submodules

catkit.gen.adsorption module

class catkit.gen.adsorption.**AdsorptionSites**(*slab*, *surface_atoms=None*, *r=6*, *tol=1e-05*)

Adsorption site object.

get_adsorption_edges(*symmetric=True*, *periodic=True*)

Return the edges of adsorption sties defined as all regions with adjacent vertices.

Parameters

- **symmetric** (*bool*) – Return only the symmetrically reduced edges.
- **periodic** (*bool*) – Return edges which are unique via periodicity.

Returns **edges** – All edges crossing ridge or vertices indexed by the expanded unit slab.

Return type ndarray (n, 2)

get_adsorption_vectors(*unique=True*, *screen=True*)

Returns the vectors representing the furthest distance from the neighboring atoms.

Returns **vectors** – Adsorption vectors for surface sites.

Return type ndarray (n, 3)

get_connectivity(*unique=True*)

Return the number of connections associated with each site.

get_coordinates(*unique=True*)

Return the 3D coordinates associated with each site.

get_periodic_sites(*screen=True*)

Return an index of the coordinates which are unique by periodic boundary conditions.

Parameters **screen** (*bool*) – Return only sites inside the unit cell.

Returns **periodic_match** – Indices of the coordinates which are identical by periodic boundary conditions.

Return type ndarray (n,)

get_symmetric_sites(*unique=True*, *screen=True*)
Determine the symmetrically unique adsorption sites from a list of fractional coordinates.

Parameters

- **unique** (*bool*) – Return only the unique symmetrically reduced sites.
- **screen** (*bool*) – Return only sites inside the unit cell.

Returns **sites** – Dictionary of sites containing index of site

Return type dict of lists

get_topology(*unique=True*)
Return the indices of adjacent surface atoms.

plot(*savefile=None*)
Create a visualization of the sites.

class catkit.gen.adsorption.**Builder**(*slab*, *surface_atoms=None*, *r=6*, *tol=1e-05*)
Bases: *catkit.gen.adsorption.AdsorptionSites*

Initial module for creation of 3D slab structures with attached adsorbates.

add_adsorbate(*adsorbate*, *bonds=None*, *index=0*, ***kwargs*)
Add and adsorbate to a slab.

Parameters

- **adsorbate** (*gratoms object*) – Molecule to connect to the surface.
- **bonds** (*int or list of 2 int*) – Index of adsorbate atoms to be bonded.
- **index** (*int*) – Index of the site or edge to use as the adsorption position. A value of -1 will return all possible structures.

Returns **slabs** – Slab(s) with adsorbate attached.

Return type gratoms object

add_adsorbates(*adsorbates*, *indices*)

ex_sites(*index*, *select='inner'*, *cutoff=0*)
Get site indices inside or outside of a cutoff radii from a provided periodic site index. If two sites are provided, an option to return the mutually inclusive points is also available.

catkit.gen.adsorption.**get_adsorption_sites**(*slab*, *surface_atoms=None*, *symmetry_reduced=True*, *tol=1e-05*)
Get the adsorption sites of a slab as defined by surface symmetries of the surface atoms.

Parameters

- **slab** (*Atoms object*) – The slab to find adsorption sites for. Must have surface atoms defined.
- **surface_atoms** (*array_like (N,)*) – List of the surface atom indices.
- **symmetry_reduced** (*bool*) – Return the symmetrically unique sites only.
- **adsorption_vectors** (*bool*) – Return the adsorption vectors.

Returns

- **coordinates** (*ndarray (N, 3)*) – Cartesian coordinates of activate sites.
- **connectivity** (*ndarray (N,)*) – Number of bonds formed for a given adsorbate.
- **symmetry_index** (*ndarray (N,)*) – Arbitrary indices of symmetric sites.

catkit.gen.geometry module

catkit.gen.molecules module

catkit.gen.molecules.**bin_hydrogen**(*hydrogens=1*, *bins=1*)
Recursive function for determining distributions of hydrogens across bins

catkit.gen.molecules.**get_topologies**(*symbols*, *saturate=False*)
Return the possible topologies of a given chemical species

Parameters

- **symbols** (*str*) – Atomic symbols to construct the topologies from.
- **saturate** (*bool*) – Saturate the molecule with hydrogen based on the default.radicals set.

catkit.gen.molecules.**hydrogenate**(*atoms*, *bins*, *copy=True*)
Add hydrogens to a gratoms object via provided bins

catkit.gen.pathways module

class catkit.gen.pathways.**ReactionNetwork**(*db_name='reaction-network.db'*, *base_valence=None*, *nbond_limits=None*)
A class for accessing a temporary SQLite database. This function works as a context manager and should be used as follows:

with ReactionNetwork() as rn: (Perform operation here)

This syntax will automatically construct the temporary database, or access an existing one. Upon exiting the indentation, the changes to the database will be automatically committed.

create_table()
Create the SQLite database table framework.

load_3d_structures(*ids=None*)
Return Gratoms objects from the ReactionNetwork database.

Parameters **ids** (*int or list of int*) – Identifier of the molecule in the database. If None, return all structure.

Returns **images** – All Gratoms objects in the database.

Return type list

load_molecules(*ids=None*, *binned=False*)
Load 2D molecule graphs from the database.

Parameters **binned** (*bool*) – Return the molecules in sub-dictionaries of their corresponding composition and bonding tags.

Returns **molecules** – All molecules present in the database.

Return type dict

load_pathways(*broken_bonds=False*)
Save enumerated pathways the ReactionNetwork database. More than two reactants or two products is not supported.

Parameters **broken_bonds** (*bool*) – Return the index information of which bond was broken. Only supported for elementary steps.

Returns **pathways** – All pathways present in the database.

Return type list

molecule_search(*element_pool={'C': 2, 'H': 6}*, *load_molecules=True*, *multiple_bond_search=False*)
Return the enumeration of molecules which can be produced from the specified atoms.

Parameters

- **element_pool** (*dict*) – Atomic symbols keys paired with the maximum number of that atom.
- **load_molecules** (*bool*) – Load any existing molecules from the database.
- **multiple_bond_search** (*bool*) – Allow atoms to form bonds with other atoms in the molecule.

path_search(*reconfiguration=True*, *substitution=True*)
Search for reaction mechanisms from a pre-populated database of molecules. By default, only single bond addition pathways are enumerated (Also called elementary steps).

Parameters

- **reconfiguration** (*bool*) – Search for reconfiguration paths. Reconfiguration paths are all those where only the bond order is changed. R1 –> P1.
- **substitution** (*bool*) – Search for substitution paths. Substitution paths are all those where one bond is broken and one bond is formed simultaneously. R1 + R2 –> P1 + P2.

plot_reaction_network(*file_name=None*)
Plot the reaction network present in the database.

save_3d_structure(*gratoms*, *overwrite=False*)
Save Cartesian coordinates into the ReactionNetwork database.

Parameters

- **gratoms** (*Atoms object*) – Structure containing Cartesian coordinates to be saved.
- **overwrite** (*bool*) – Allow the database to overwrite a matching index.

save_molecules(*molecules*)
Save enumerated molecules to the ReactionNetwork database.

Parameters **molecules** (*dict*) – Molecules to be saved to the database.

save_pathways(*pathways*, *broken_bonds=None*)
Save enumerated pathways the ReactionNetwork database. More than two reactants or two products is not supported.

Parameters

- **pathways** (*list*) – Sorted pathways in the form [R1, R2, P1, P2].
- **broken_bonds** (*list*) – Comma separated strings of index associated with the two atoms whos bond is broken. List order must match pathways.

catkit.gen.route module

catkit.gen.route.**get_heppel_sellers**(*nu*, *terminal*)
Returns an array of linearly independent reaction routes as described by Heppel-Sellers reaction route enumeration.

Parameters

- **nu** (*ndarray (n, m)*) – The stoichiometric matrix of n species and m mechanisms.
- **terminal** (*ndarray (j,)*) – Indices of the m species to be considered as terminal

Returns **sigma** – Linearly independent set of Heppel-Sellers reaction routes.

Return type ndarray (m, k)

catkit.gen.route.**get_reaction_routes**(*nu*, *sigma*, *empty_routes=True*, *independent_only=False*)

Returns an array of reaction routes. Returns all full reaction routes by default.

Parameters

- **nu** (*ndarray (n, m)*) – The stoichiometric matrix of n species and m mechanisms.
- **sigma** (*ndarray (m, j)*) – A linearly independent set of reaction routes.
- **empty_routes** (*bool*) – Return the empty routes along with the full routes.
- **independent_only** (*bool*) – Return only a linearly independent set of full reaction routes. Can take less time.

Returns

- **FR** (*ndarray (m, k)*) – Enumerated full reaction routes.
- **ER** (*ndarray (m, l)*) – Enumerated empty reaction routes.

catkit.gen.route.**get_response_reactions**(*epsilon*, *selection=None*, *species=False*)

Returns an array of possible response reaction routes for a given chemical formula array.

Parameters

- **epsilon** (*ndarray (n, m)*) – Chemical formula array of n elements by m molecular species.
- **selection** (*ndarray (j,)*) – Indices of the m species to be considered as terminal
- **species** (*bool*) – Return the indices of the chemical species used.

Returns

- **RER** (*ndarray (m, k)*) – Possible response reaction routes.
- **index** (*ndarray (j, k)*) – Indices of the k terminal species use to produce the l response reactions.

catkit.gen.surface module

class catkit.gen.surface.**SlabGenerator**(*bulk*, *miller_index*, *layers*, *vacuum=None*, *fixed=None*, *layer_type='ang'*, *attach_graph=True*, *standardize_bulk=False*, *primitive=True*, *tol=1e-08*)

Bases: object

Class for generation of slab unit cells from bulk unit cells.

Many surface operations rely upon / are made easier through the bulk basis cell they are created from. The SlabGenerator class is designed to house these operations.

adsorption_sites(*slab*, ***kwargs*)

Helper function to return the adsorption sites of the provided slab.

Parameters **slab** (*atoms object*) – The slab to find adsorption sites for. Assumes you are using the same basis.

Returns

- **coordinates** (*ndarray (n,)*) – Coordinates of the adsorption sites
- **connectivity** (*ndarray (n,)*) – Connectivity of the adsorption sites

align_crystal(*bulk*, *miller_index*)

Return an aligned unit cell from bulk unit cell. This alignment rotates the a and b basis vectors to be parallel to the Miller index.

Parameters

- **bulk** (*Atoms object*) – Bulk system to be standardized.
- **miller_index** (*list (3,)*) – Miller indices to align with the basis vectors.

Returns **new_bulk** – Standardized bulk unit cell.

Return type Gratoms object

get_slab(*size=1*, *iterm=0*)

Generate a slab from the bulk structure. This function is meant specifically for selection of an individual termination or enumeration through surfaces of various size.

This function will orthogonalize the c basis vector and align it with the z-axis which breaks bulk symmetry along the z-axis.

Parameters

- **size** (*int, array_like (2,) or (2, 2)*) – Size of the unit cell to create as described in *set_size()*.
- **iterm** (*int*) – A termination index in reference to the list of possible terminations.

Returns **slab** – The modified basis slab produced based on the layer specifications given.

Return type Gratoms object

get_slab_basis(*iterm=0*, *maxn=20*)

Return a list of all terminations which have been properly shifted and with an appropriate number of layers added. This function is mainly for performance, to prevent looping over other operations which are not related the size of the slab.

This step also contains periodically constrained orthogonalization of the c basis. This implementation only works if the a anb b basis vectors are properly aligned with the x and y axis. This is strictly to assist the correct identification of surface atoms.

Only produces the terminations requested as a lazy evaluator.

Parameters

- **iterm** (*int*) – Index of the slab termination to return.
- **maxn** (*int*) – The maximum integer component to search for a more orthogonal bulk.

Returns **ibasis** – Prepared, ith basis.

Return type Gratoms object

get_unique_terminations()

Determine the fractional coordinate shift that will result in a unique surface termination. This is not required if bulk standardization has been performed, since all available z shifts will result in a unique termination for a primitive cell.

Returns **unique_shift** – Fractional coordinate shifts which will result in unique terminations.

Return type array (n,)

make_symmetric(*slab*)
Returns a symmetric slab. Note, this will trim the slab potentially resulting in loss of stoichiometry.

set_size(*slab*, *size*)
Set the size of a slab based one of three methods.

1. An integer value performs a search of valid matrix operations to perform on the ab-basis vectors to return a set which with a minimal sum of distances and an angle closest to 90 degrees.

2. An array_like of length 2 will multiply the existing basis vectors by that amount.

3. An array of shape (2, 2) will be interpreted as matrix notation to multiply the ab-basis vectors by.

Parameters

- **slab** (*Atoms object*) – Slab to be made into the requested size.
- **size** (*int, array_like (2,) or (2, 2)*) – Size of the unit cell to create as described above.

Returns **supercell** – Supercell of the requested size.

Return type Gratoms object

catkit.gen.surface.**convert_miller_index**(*miller_index*, *atoms1*, *atoms2*)
Return a converted miller index between two atoms objects.

catkit.gen.surface.**generate_indices**(*max_index*)
Return an array of miller indices enumerated up to values plus or minus some maximum. Filters out lists with greatest common divisors greater than one. Only positive values need to be considered for the first index.

Parameters **max_index** (*int*) – Maximum number that will be considered for a given surface.

Returns **unique_index** – Unique miller indices

Return type ndarray (n, 3)

catkit.gen.surface.**get_degenerate_indices**(*bulk*, *miller_index*)
Return the miller indices which are degenerate to a given miller index for a particular bulk structure.

Parameters

- **bulk** (*Atoms object*) – Bulk structure to get the degenerate miller indices.
- **miller_index** (*array_like (3,)*) – Miller index to get the degenerate indices for.

Returns **degenerate_indices** – Degenerate miller indices to the provided index.

Return type array (N, 3)

catkit.gen.surface.**get_unique_indices**(*bulk*, *max_index*)
Returns an array of miller indices which will produce unique surface terminations based on a provided bulk structure.

Parameters

- **bulk** (*Atoms object*) – Bulk structure to get the unique miller indices.
- **max_index** (*int*) – Maximum number that will be considered for a given surface.

Returns **unique_millers** – Symmetrically distinct miller indices for a given bulk.

Return type ndarray (n, 3)

catkit.gen.surface.**transform_ab**(*slab*, *matrix*, *tol=1e-05*)

Transform the slab basis vectors parallel to the z-plane by matrix notation. This can result in changing the slabs cell size. This can also result in very unusual slab dimensions, use with caution.

Parameters

- **slab** (*Atoms object*) – The slab to be transformed.
- **matrix** (*array_like (2, 2)*) – The matrix notation transformation of the a and b basis vectors.
- **tol** (*float*) – Float point precision tolerance.

Returns **slab** – Slab after transformation.

Return type Atoms object

catkit.gen.utils module

catkit.gen.utils.**get_voronoi_neighbors**(*atoms*, *r=4*)

Return the connectivity matrix from the Voronoi method. Multi-bonding occurs through periodic boundary conditions.

Parameters

- **atoms** (*atoms object*) – Atoms object with the periodic boundary conditions and unit cell information to use.
- **r** (*float*) – Radius of the spheres to expand around each atom.

Returns **connectivity** – Number of edges formed between atoms in a system.

Return type ndarray (n, n)

catkit.gen.utils.**get_cutoff_neighbors**(*atoms*, *cutoff=None*, *atol=1e-08*)

Return the connectivity matrix from a simple radial cutoff. Multi-bonding occurs through periodic boundary conditions.

Parameters

- **atoms** (*atoms object*) – Atoms object with the periodic boundary conditions and unit cell information to use.
- **cutoff** (*float*) – Cutoff radius to use when determining neighbors.
- **atol** (*float*) – Absolute tolerance to use when computing distances.

Returns **connectivity** – Number of edges formed between atoms in a system.

Return type ndarray (n, n)

catkit.gen.utils.**trilaterate**(*centers*, *r*, *zvector=None*)

Find the intersection of two or three spheres. In the case of two sphere intersection, the z-coordinate is assumed to be an intersection of a plane whose normal is aligned with the points and perpendicular to the positive z-coordinate.

If more than three spheres are supplied, the centroid of the points is returned (no radii used).

Parameters

- **centers** (*list | ndarray (n, 3)*) – Cartesian coordinates representing the center of each sphere
- **r** (*list | ndarray (n,)*) – The radii of the spheres.

- **zvector** (*ndarray (3,)*) – The vector associated with the upward direction for underspecified coordinations (1 and 2).

Returns **intersection** – The point where all spheres/planes intersect.

Return type ndarray (3,)

catkit.gen.utils.**get_unique_xy**(*xyz_coords*, *cutoff=0.1*)

Return the unique coordinates of an atoms object for the requrested atoms indices. Z-coordinates are projected to maximum z-coordinate by default.

Parameters

- **xyz_coords** (*ndarray (n, 3)*) – Cartesian coordinates to identify unique xy positions from.
- **cutoff** (*float*) – Distance in Angstrons to consider xy-coordinate unique within.

Returns **xy_pos** – Unique xy coordinates projected onto a maximal z coordinate.

Return type ndarray (m, 3)

catkit.gen.utils.**expand_cell**(*atoms*, *r=6*)

Return Cartesian coordinates atoms within a supercell which contains spheres of specified cutoff radius around all atom positions.

Parameters

- **atoms** (*Atoms object*) – Atoms with the periodic boundary conditions and unit cell information to use.
- **r** (*float*) – Radius of the spheres to expand around each atom.

Returns

- **index** (*ndarray of int*) – Indices associated with the original unit cell positions.
- **coords** (*ndarray of (3,) array*) – Cartesian coordinates associated with positions in the supercell.

catkit.gen.utils.**matching_sites**(*position*, *comparators*, *tol=1e-08*)

Get the indices of all points in a comparator list that are equal to a given position (with a tolerance), taking into account periodic boundary conditions (adaptation from Pymatgen).

This will only accept a fractional coordinate scheme.

Parameters

- **position** (*list (3,)*) – Fractional coordinate to compare to list.
- **comparators** (*list (3, n)*) – Fractional coordinates to compare against.
- **tol** (*float*) – Absolute tolerance.

Returns **match** – Indices of matches.

Return type list (n,)

catkit.gen.utils.**connectivity_to_edges**(*connectivity*, *indices=None*)

Convert a Numpy connectivity matrix into a list of NetworkX compatible edges.

catkit.gen.utils.**isomorphic_molecules**(*graph0*, *graph1*)

Check whether two molecule graphs are isomorphic.

catkit.gen.utils.**get_spglib_cell**(*atoms*, *primitive=False*, *idealize=True*, *tol=1e-05*)

Atoms object interface with spglib primitive cell finder: https://atztogo.github.io/spglib/python-spglib.html#python-spglib

Parameters

- **atoms** (*object*) – Atoms object to search for a primitive unit cell.
- **primitive** (*bool*) – Reduce the atoms object into a primitive form.
- **idealize** (*bool*) – Convert the cell into the spglib standardized form.
- **tol** (*float*) – Tolerance for floating point rounding errors.

Returns **primitive cell** – The primitive unit cell returned by spglib if one is found.

Return type object

catkit.gen.utils.**get_point_group**(*atoms*, *tol=1e-08*)
Return the point group operations of a systems.

catkit.gen.utils.**get_symmetry**(*atoms*, *tol=1e-05*)
Atoms object interface with spglib symmetry finder: https://atztogo.github.io/spglib/python-spglib.html#python-spglib

Parameters

- **atoms** (*object*) – Atoms object to search for symmetric structures of.
- **tol** (*float*) – Tolerance for floating point rounding errors.

Returns **symmetry operations** – Symmetry operations from spglib.

Return type ndarray (n, n)

catkit.gen.utils.**get_affine_operations**(*atoms*)
Return the affine operations of a unit cell.

catkit.gen.utils.**matching_coordinates**(*position*, *comparators*, *tol=1e-08*)
Get the indices of all points in a comparator list that are equal to a given position (with a tolerance), taking into account periodic boundary conditions (adaptation from Pymatgen).

This will only accept a Cartesian coordinate scheme. TODO: merge this with matching_sites.

Parameters

- **position** (*list (3,)*) – Fractional coordinate to compare to list.
- **comparators** (*list (3, n)*) – Fractional coordinates to compare against.
- **tol** (*float*) – Absolute tolerance.

Returns **match** – Indices of matches.

Return type list (n,)

catkit.gen.utils.**get_unique_coordinates**(*atoms*, *axis=2*, *tag=False*, *tol=0.001*)
Return unique coordinate values of a given atoms object for a specified axis.

Parameters

- **atoms** (*object*) – Atoms object to search for unique values along.
- **axis** (*int (0, 1, or 2)*) – Look for unique values along the x, y, or z axis.
- **tag** (*bool*) – Assign ASE-like tags to each layer of the slab.
- **tol** (*float*) – The tolerance to search for unique values within.

Returns **values** – Array of unique positions in fractional coordinates.

Return type ndarray (n,)

catkit.gen.utils.**get_reciprocal_vectors**(*atoms*)
Return the reciprocal lattice vectors to a atoms unit cell.

catkit.gen.utils.**plane_normal**(*xyz*)
Return the surface normal vector to a plane of best fit.

Parameters **xyz** (*ndarray (n, 3)*) – 3D points to fit plane to.

Returns **vec** – Unit vector normal to the plane of best fit.

Return type ndarray (1, 3)

catkit.gen.utils.**running_mean**(*array*, *N=5*)
Calculate the running mean of array for N instances.

Parameters

- **array** (*array_like | ndarray (N,)*) – Array of values to have a average taken from.
- **N** (*int*) – Number of values to take an average with.

Returns **running_mean** – Mean value of the running average.

Return type ndarray (N + 1,)

catkit.gen.utils.**to_gratoms**(*atoms*)
Convert and atom object to a gratoms object.

catkit.gen.utils.**get_atomic_numbers**(*formula*, *return_count=False*)
Return the atomic numbers associated with a chemical formula.

Parameters

- **formula** (*string*) – A chemical formula to parse into atomic numbers.
- **return_count** (*bool*) – Return the count of each element in the formula.

Returns

- **numbers** (*ndarray (n,)*) – Element numbers in associated species.
- **counts** (*ndarray (n,)*) – Count of each element in a species.

catkit.gen.utils.**get_reference_energies**(*species*, *energies*)
Get reference energies for the elements in a set of molecules.

Parameters

- **species** (*list (n,)*) – Chemical formulas for each molecular species.
- **energies** (*list (n,)*) – Total energies associated with each species.

Returns

- **elements** (*ndarray (n,)*) – Atomic elements associated with all species.
- **references** (*ndarray (n,)*) – Reference energies associated with each element.

catkit.gen.utils.**parse_slice**(*slice_name*)
Return a correctly parsed slice from input of varying types.

catkit.gen.utils.**ext_gcd**(*a*, *b*)
Extension of greatest common divisor.

catkit.gen.utils.**list_gcd**(*values*)
Return the greatest common divisor of a list of values.

Module contents

Catalysis Generator.

class catkit.gen.**Defaults**
Bases: _abcoll.MutableMapping, dict

No frills default dictionary class.

catkit.pawprint package

Submodules

catkit.pawprint.generator module

class catkit.pawprint.generator.**Fingerprinter**(*images=None*)
Parent class for all fingerprint generators.

get_fp(*parameters*, *operation_list*)
Return the fingerprints for a list of images of single atoms object for the given parameters.

Parameters

- **parameters** (*list of str*) – Names of seeding parameters available in the parameters database.
- **operation_list** (*list of functions*) – A list of operation functions to produce the fingerprints from.

Returns **fingerprints** – Fingerprints for the images produced from the provided seed parameters.

Return type ndarray (n, m)

catkit.pawprint.generator.**get_connectivity**(*atoms*, *method=None*)
Returns an estimate of the connectivity matrix for a given atoms-object from CatGen.

Parameters

- **atoms** (*object*) – Molecular structure with out without adsorbates.
- **method** (*str (None or 'voronoi')*) – Method for estimating the connectivity matrix:

  None - standard cutoff radius method. voronoi - best suited for bulk characterization.

Returns **connectivity** – Estimated connectivity matrix where n is the number of atoms in the atoms-object.

Return type ndarray (n, n)

catkit.pawprint.operations module

catkit.pawprint.operations.**autocorrelation**(*atoms=None*, *atoms_parameters=None*, *connectivity=None*, *d=0*)
Autocorrelation convolution for systems without pbc.

catkit.pawprint.operations.**bonding_convolution**(*atoms=None*, *atoms_parameters=None*, *connectivity=None*)
Perform convolution of metal atoms with bonded adsorbates.

catkit.pawprint.operations.**periodic_convolution**(*atoms=None*, *atoms_parameters=None*, *connectivity=None*, *d=0*, *normalize=False*)
Return the square of each property with each atom. For distance 1 the covolutions returns the multiple of each property for all neighboring atom pairs.

catkit.pawprint.operations.**raw_properties**(*atoms=None*, *atoms_parameters=None*, *connectivity=None*)
Return all atom properties without manipulation.

Module contents

class catkit.pawprint.**Fingerprinter**(*images=None*)
Parent class for all fingerprint generators.

get_fp(*parameters*, *operation_list*)
Return the fingerprints for a list of images of single atoms object for the given parameters.

Parameters

- **parameters** (*list of str*) – Names of seeding parameters available in the parameters database.
- **operation_list** (*list of functions*) – A list of operation functions to produce the fingerprints from.

Returns **fingerprints** – Fingerprints for the images produced from the provided seed parameters.

Return type ndarray (n, m)

catkit.flow package

Submodules

catkit.flow.fwespresso module

catkit.flow.fwespresso.**get_neb**(*in_file='input.traj'*)
Performs a ASE NEB optimization with the ase-espresso calculator with the keywords defined inside the atoms object information.

Parameters **in_file** (*str*) – Name of the input file to load from the local directory.

catkit.flow.fwespresso.**get_pdos**(*out_file='dos.msg'*)
Calculate and save the projected DOS. Requires a previously relaxed calculation.

Parameters **out_file** (*str*) – Name of the output file to save the results to.

catkit.flow.fwespresso.**get_potential_energy**(*in_file='input.traj'*)
Performs a ASE get_potential_energy() call with the ase-espresso calculator with the keywords defined inside the atoms object information.

This can be a singlepoint calculation or a full relaxation depending on the keywords.

Parameters **in_file** (*str*) – Name of the input file to load from the local directory.

catkit.flow.fwespresso.**get_relaxed_calculation**(*in_file='output.traj'*)
Attach a stored calculator in the current directory to the provided atoms object.

Then return the atoms object with the calculator attached.

Parameters **in_file** (*str*) – Name of the relaxed trajectroy file to load.

catkit.flow.fwespresso.**get_total_potential**(*out_file='potential.msg'*)
Calculate and save the total potential. Requires a previously relaxed calculation.

Parameters **out_file** (*str*) – Name of the output file to save the results to.

catkit.flow.fwio module

catkit.flow.fwio.**array_to_list**(*data*)
A function to covert all arrays in a structure of embeded dictionaries and lists into lists themselves.

catkit.flow.fwio.**atoms_to_encode**(*images*)
Converts an list of atoms objects to an encoding from a .traj file.

catkit.flow.fwio.**encode_to_atoms**(*encode*, *out_file='input.traj'*)
Dump the encoding to a local traj file.

catkit.flow.hpcio module

catkit.flow.hpcio.**get_nnodes**()
Get the number of nodes being used in this environment. This is meant to be run on a calculation node.

catkit.flow.hpcio.**get_server**()
A function for determining which server the job is currently running on. This is meant to be run on a calculation node.

catkit.flow.qeio module

catkit.flow.qeio.**attach_results**(*f*, *atoms*, *write_file=True*)
Return the TS corrected energy for a scf instance in a log file and attach them to the given atoms obejct.

Will also attach the forces and stress if applicable.

catkit.flow.qeio.**cd**(**args*, ***kwds*)
Does path management: if the path doesn't exists, create it otherwise, move into it until the indentation is broken.

e.g.

with cd('the/path/is/real'): 'do things in the new path'

Parameters **path** (*str*) – Directory path to create and change into.

catkit.flow.qeio.**geometry_hash**(*atoms*)
A hash based strictly on the geometry features of an atoms object: positions, cell, and symbols.

This is intended for planewave basis set calculations, so pbc is not considered.

Each element is sorted in the algorithem to help prevent new hashs for identical geometries.

catkit.flow.qeio.**log_to_atoms**(*log_file='log'*, *ent=-1*, *out_file=None*)
Parse a QE log file for atoms trajectory and return a list of atoms objects representative of the relaxation path.

NOTE: trajectory information is only returned for calculations run with BFGS internal to QE.

catkit.flow.qeio.**write_to_db**(*path*, *db_name='master.db'*, *keys={}*, *traj=False*, *pdos=False*)
This function is used to write an ASE database entry for each atoms image inside a QE directory.

Module contents

catkit.hub package

Subpackages

catkit.hub.ase_tools package

Submodules

catkit.hub.ase_tools.gas_phase_references module

catkit.hub.ase_tools.gas_phase_references.**construct_reference_system**(*symbols*, *candidates=['H2', 'H2O', 'NH3', 'CH4', 'CO', 'SH2', 'HCl', 'N2', 'O2']*)

Take a list of symbols and construct gas phase references system, when possible avoiding O2. Candidates can be rearranged, where earlier candidates get higher preference than later candidates

assume symbols sorted by atomic number

catkit.hub.ase_tools.gas_phase_references.**get_atomic_stoichiometry**(*references*)

Given a list of references (tuples of (symbol, molecule)) return stoichiometry matrix that connects atomic symbols to its molecular references.

catkit.hub.ase_tools.gas_phase_references.**get_stoichiometry_factors**(*adsorbates*, *references*)

Take a list of adsorabtes and a corresponding reference system and return a list of dictionaries encoding the stoichiometry factors converting between adsorbates and reference molecules.

catkit.hub.ase_tools.gas_phase_references.**molecules2symbols**(*molecules*, *add_hydrogen=True*)

Take a list of molecules and return just a list of atomic symbols, possibly adding hydrogen

Module contents

catkit.hub.ase_tools.**check_in_ase**(*filename*, *ase_db*, *energy=None*)

Check if entry is allready in ASE db

catkit.hub.ase_tools.**check_traj**(*filename*, *strict=True*, *verbose=True*)

catkit.hub.ase_tools.**clear_prefactor**(*molecule*)

catkit.hub.ase_tools.**clear_state**(*name*)

catkit.hub.ase_tools.**debug_assert**(*expression*, *message*, *debug=False*)

catkit.hub.ase_tools.**get_atomic_numbers**(*filename*)

catkit.hub.ase_tools.**get_atoms**(*molecule*)

catkit.hub.ase_tools.**get_bulk_composition**(*filename*)

catkit.hub.ase_tools.**get_chemical_formula**(*filename*, *mode='metal'*)

catkit.hub.ase_tools.**get_energies**(*filenames*)

catkit.hub.ase_tools.**get_energy**(*filename*)

catkit.hub.ase_tools.**get_energy_diff**(*filename*, *filename_ref*)

catkit.hub.ase_tools.**get_formula_from_numbers**(*numbers*)

catkit.hub.ase_tools.**get_layers**(*atoms*)

catkit.hub.ase_tools.**get_n_layers**(*filename*)

catkit.hub.ase_tools.**get_number_of_atoms**(*filename*)

catkit.hub.ase_tools.**get_numbers_from_formula**(*formula*)

catkit.hub.ase_tools.**get_pbc**(*filename*)

catkit.hub.ase_tools.**get_reaction_atoms**(*reaction*)

catkit.hub.ase_tools.**get_reaction_energy**(*traj_files*, *reaction*, *reaction_atoms*, *states*, *prefactors*, *prefactors_TS*, *energy_corrections*)

catkit.hub.ase_tools.**get_reaction_from_folder**(*folder_name*)

catkit.hub.ase_tools.**get_reference**(*filename*)

catkit.hub.ase_tools.**get_state**(*name*)

catkit.hub.ase_tools.**get_surface_composition**(*filename*)

catkit.hub.ase_tools.**get_traj_str**(*filename*)

catkit.hub.ase_tools.**read_ase**(*filename*)

catkit.hub.ase_tools.**tag_atoms**(*atoms*, *types=None*)

catkit.hub.ase_tools.**update_ase**(*db_file*, *identity*, ***key_value_pairs*)
Connect to ASE db

catkit.hub.ase_tools.**write_ase**(*filename*, *db_file*, *user=None*, *data=None*, ***key_value_pairs*)
Connect to ASE db

Submodules

catkit.hub.ase_connect module

catkit.hub.ase_connect.**main**()

catkit.hub.cathubsqlite module

class catkit.hub.cathubsqlite.**CathubSQLite**(*filename*)
Class for managing SQLite3 database for reaction energies, publications and atomic structures. Builds on top of the ASE database for atomic strucutres https://wiki.fysik.dtu.dk/ase/ase/db/db.html with four additional tables:

publication: publication info

publication_system: one-to-many mapping between publication table and systems table in ASE database

reaction: reaction energies for surfaces

reaction_system: mamy-to-many mapping between reaction table and systems table in ASE database

Connect to a database object:

db = CathubSQLite('yourdbfile.db')

Set up a connection for several manipulations:

with db as CathubSQLite('yourdbfile.db'): Do your work...

Parameters **filename** (*str*) – name of database file

check(*chemical_composition*, *reaction_energy*)
Check if entry with same surface and energy is allready written to database file :param chemcial_composition: :type chemcial_composition: str :param reaction_energy: :type reaction_energy: str :param Returns id or None:

check_publication(*pub_id*)

check_publication_structure(*pub_id*, *ase_id*)

check_reaction_on_surface(*chemical_composition*, *reactants*, *products*)
Check if entry with same surface and reaction is allready written to database file

Parameters

- **chemcial_composition** (*str*) –
- **reactants** (*dict*) –
- **products** (*dict*) –
- **id or None** (*Returns*) –

get_last_id(*cur*, *table='reaction'*)
Get the id of the last written row in table

Parameters

- **cur** (*database connection().cursor() object*) –
- **table** (*str*) – 'reaction', 'publication', 'publication_system', 'reaction_system'
- **Returns** (*id*) –

read(*id*, *table='reaction'*)
Return an entire row of a table :param id: row integer :type id: int :param table: 'reaction', 'publication', 'publication_system', 'reaction_system' :type table: str

update(*id*, *values*, *key_names='all'*)
Update reaction info for a selected row

Parameters

- **id** (*int*) – row integer
- **values** (*dict*) – See write() method for details
- **key_names** (*list or 'all'*) – list with name of columns to update. Should match the keys-value pairs in values. default is 'all'

write(*values*, *data=None*)
Write reaction info to db file

Parameters

- **values** (*dict*) –
- **values dict can include** (*The*) –
- **{'chemical_composition'** (*str (chemical composition on empty slab) ,*) –
- **'surface_composition'** (*str (reduced chemical composition or*) – shortname),
- **'facet'** (*str*) –
- **'sites'** (*dict*) – adsorption sites of species. f.ex: {'OH': 'ontop', 'O': 'hollow'}
- **'coverages'** (*dict*) – coverage of adsorbates relative to the unit cell f.ex. {'OH': 0.25, 'O': 0.5})
- **'products'** (*'reactants'/*) – keys with name of chemical species folloved by phase (gas, *) values are the prefactor in the reaction. For reaction H2Ogas -> 2Hstar + O star you would write: 'reactants': {OHstar: 1, Hstar: 2} 'products': {OHstar: 1, Hstar: 2}
- **'reaction_energy'** (*float*) –
- **'activation_energy'** (*float*) –
- **'dft_code'** (*str*) –
- **'dft_functional'** (*str*) –
- **'username'** (*str*) –
- **'pub_id'** (*str*) – Should match the pub_id of the corresponding publications
- **}** –

write_publication(*values*)
Write publication info to db

Parameters **values** (*dict with entries*) – {'pub_id': str (short name for publication), 'authors': list of str () 'journal': str, 'volume': str, 'number': str, 'pages': 'str' 'year': int, 'publisher': str, 'doi': str, 'tags': list of str}

catkit.hub.cathubsqlite.**check_ase_ids**(*values*, *ase_ids*)

catkit.hub.cathubsqlite.**get_key_value_list**(*key_list*, *values*, *table='reaction'*)

catkit.hub.cathubsqlite.**get_key_value_str**(*values*)

catkit.hub.cathubsqlite.**get_value_strlist**(*value_list*)

catkit.hub.cli module

catkit.hub.convert_traj module

catkit.hub.convert_traj.**main**(*base*)

catkit.hub.create_user module

catkit.hub.create_user.**main**(*user*)

catkit.hub.db2server module

catkit.hub.db2server.**main**(*dbfile*, *start_id=1*, *write_reaction=True*, *write_ase=True*, *write_publication=True*, *write_reaction_system=True*, *block_size=1000*, *start_block=0*, *db_user='catroot'*, *db_password=None*)

catkit.hub.folder2db module

catkit.hub.folder2db.**main**(*folder_name*, *debug=False*, *skip=[]*, *goto_reaction=None*, *old=False*)

catkit.hub.folder_check module

catkit.hub.folder_check.**main**(*folder*)

catkit.hub.folderreader module

class catkit.hub.folderreader.**FolderReader**(*folder_name*, *debug=False*, *strict=True*, *verbose=False*, *update=True*)

Class for reading data from organized folders and writing to local CathubSQLite database. Folders should be arranged with make_folders_template and are read in the order:

level:

0 folder_name 1 |– publication 2 |– dft_code 3 |– dft_functional 4 |– gas 4 |– metal1 5 |– facet 6 |– reaction

Parameters

- **foldername** (*str*) –
- **debug** (*bool*) – default is False. Choose True if the folderreader should continue in spite of errors.
- **update** (*bool*) – Update data if allready present in database file. defalt is True

read(*skip=[]*, *goto_metal=None*, *goto_reaction=None*)

Get reactions from folders.

Parameters

- **skip** (*list of str*) – list of folders not to read
- **goto_reaction** (*str*) – Skip ahead to this metal
- **goto_reaction** – Skip ahead to this reacion

read_bulk(*root*, *files*)

read_energies(*root*, *files*)

read_gas(*root*)

read_pub(*root*)

read_reaction(*root*, *files*)

read_slab(*root*, *files*)

write(*skip=[]*, *goto_reaction=None*)

write_publication(*pub_data*)

catkit.hub.folderreader.**read_name_from_folder**(*root*)

catkit.hub.maintain_server module

class catkit.hub.maintain_server.**MaintainPostgres**(*user='catroot'*, *password=None*, *stdin=<open file '<stdin>'*, *mode 'r'>*, *stdout=<open file '<stdout>'*, *mode 'w'>*)

Bases: *catkit.hub.postgresql.CathubPostgreSQL*

delete_lost_systems()

fill_reaction_system()

catkit.hub.make_folders_template module

catkit.hub.organize module

catkit.hub.organize.**collect_structures**(*foldername*, *options*)

catkit.hub.organize.**create_folders**(*options*, *structures*, *root=''*)

catkit.hub.organize.**fuzzy_match**(*structures*, *options*)

catkit.hub.organize.**get_chemical_formula**(*atoms*)

Compatibility function, return mode=metal, when available, mode=hill, when not (ASE <= 3.13)

catkit.hub.organize.**main**(*options*)

catkit.hub.organize.**symbols**(*atoms*)

catkit.hub.postgresql module

class catkit.hub.postgresql.**CathubPostgreSQL**(*user='catroot'*, *password=None*, *stdin=<open file '<stdin>'*, *mode 'r'>*, *stdout=<open file '<stdout>'*, *mode 'w'>*)

Class for setting up the catalysis hub reaction energy database on postgreSQL server.

check(*pub_id*, *chemical_composition*, *reactants*, *products*, *reaction_energy=None*, *strict=True*)

create_user(*user*)

delete(*authorlist*, *year*, *doi=None*)

publication_status()

read(*id*, *table='reaction'*)

remove_user(*user*)

status(*table='reaction'*)

transfer(*filename_sqlite*, *start_id=1*, *write_ase=True*, *write_publication=True*, *write_reaction=True*, *write_reaction_system=True*, *block_size=1000*, *start_block=0*)

update(*id*, *values*, *key_names='all'*)

update_publication(*pub_dict*)

write(*values*, *table='reaction'*)

write_publication(*pub_values*)

catkit.hub.postgresql.**get_key_value_str**(*values*, *table='reaction'*)

catkit.hub.psql_server_connect module

catkit.hub.psql_server_connect.**main**(*user*)

catkit.hub.query module

catkit.hub.query.**convert**(*name*)

catkit.hub.query.**execute_graphQL**(*query_string*)

catkit.hub.query.**get_ase_db**()

catkit.hub.query.**get_atoms_by_id**(*unique_id*)

catkit.hub.query.**get_atomsrow_by_id**(*unique_id*)

catkit.hub.query.**get_publications**(***kwargs*)

catkit.hub.query.**get_reactions**(*n_results=20*, *write_db=False*, ***kwargs*)
Get reactions from server

Give key value strings as arguments

catkit.hub.query.**graphql_query**(*table='reactions'*, *subtables=[]*, *columns=['chemicalComposition'*, *'reactants'*, *'products']*, *n_results=10*, *queries={}*)

catkit.hub.query.**map_column_names**(*column*)

catkit.hub.query.**query**(*table='reactions'*, *columns=['chemicalComposition'*, *'reactants'*, *'products']*, *subtables=[]*, *n_results=10*, *queries={}*, *print_output=False*)

catkit.hub.tools module

catkit.hub.tools.**add_atoms**(*atoms_list*)

catkit.hub.tools.**check_reaction**(*reactants*, *products*)
Check the stoichiometry and format of chemical reaction used for folder structure. list of reactants -> list of products

catkit.hub.tools.**extract_atoms**(*molecule*)
Return a string with all atoms in molecule

catkit.hub.tools.**get_bases**(*folder_name*)

catkit.hub.tools.**get_catbase**()

Module contents

1.1.2 catkit.build module

catkit.build.**add_adsorbate**(*atoms*)
Add an adsorbate to a surface.

catkit.build.**bulk**(*name*, *crystalstructure=None*, *primitive=False*, ***kwargs*)
Return the standard conventional cell of a bulk structure created using ASE. Accepts all keyword arguments for the ase bulk generator.

Parameters

- **name** (*Atoms | str*) – Chemical symbol or symbols as in 'MgO' or 'NaCl'.
- **crystalstructure** (*str*) – Must be one of sc, fcc, bcc, hcp, diamond, zincblende, rocksalt, cesiumchloride, fluorite or wurtzite.
- **primitive** (*bool*) – Return the primitive unit cell instead of the conventional standard cell.

Returns **standardized_bulk** – The conventional standard or primitive bulk structure.

Return type Gratoms object

catkit.build.**molecule**(*species*, *topology=None*, *adsorption=False*, *vacuum=0*)
Return gas-phase molecule structures based on species and topology.

Parameters

- **species** (*str*) – The chemical symbols to construct a molecule from.
- **topology** (*int, str, or slice*) – The indices for the distinct topology produced by the generator.
- **adsorption** (*bool*) – Construct the molecule as though it were adsorbed to a surface parallel to the z-axis.
- **vacuum** (*float*) – Angstroms of vacuum to pad the molecule with.

Returns **images** – 3D structures of the requested chemical species and topologies.

Return type list of objects

catkit.build.**surface**(*elements*, *size*, *crystal='fcc'*, *miller=(1, 1, 1)*, *termination=0*, *fixed=0*, *vacuum=10*, ***kwargs*)
A helper function to return the surface associated with a given set of input parameters to the general surface generator.

Parameters

- **elements** (*str or object*) – The atomic symbol to be passed to the as bulk builder function or an atoms object representing the bulk structure to use.
- **size** (*list (3,)*) – Number of time to expand the x, y, and z primitive cell.
- **crystal** (*str*) – The bulk crystal structure to pass to the ase bulk builder.
- **miller** (*list (3,) or (4,)*) – The miller index to cleave the surface structure from. If 4 values are used, assume Miller-Bravis convention.
- **termination** (*int*) – The index associated with a specific slab termination.
- **fixed** (*int*) – Number of layers to constrain.
- **vacuum** (*float*) – Angstroms of vacuum to add to the unit cell.

Returns **slab** – Return a slab generated from the specified bulk structure.

Return type Gratoms object

1.1.3 catkit.enumeration module

catkit.enumeration.**surfaces**(*bulk*, *width*, *miller_indices=(1, 1, 1)*, *terminations=None*, *sizes=None*, *vacuum=10*, *fixed=0*, *layer_type='angs'*, ***kwargs*)

Return a list of enumerated surfaces based on symmetry properties of interest to the user. Any bulk structure provided will be standardized.

This function will take additional keyword arguments for the *catkit.gen.surface.SlabGenerator()* Class.

Parameters

- **bulk** (*str | Atoms*) – The atomic symbol to be passed to the as bulk builder function or an atoms object representing the bulk structure to use.
- **width** (*float*) – Minimum width of the slab in angstroms before trimming. Imposing symmetry requirements will reduce the width.
- **miller_indices** (*int | list (3,) | list of list (n, 3)*) – List of the miller indices to enumerate slabs for. If an integer is provided, the value is treated as the maximum miller index to consider for an enumeration of all possible unique miller indices.
- **terminations** (*int | array_like*) – Return the terminations associated with the provided indices. If -1, all possible terminations are enumerated.
- **sizes** (*None | int | array_like (n,)*) – Enumerate all surface sizes in the provided list. Sizes are integers which represent multiples of the smallest possible surface area. If None, return slabs with the smallest possible surface area. If an integer, enumerate all sizes up to that multiple.
- **vacuum** (*float*) – Angstroms of vacuum to add to the unit cell.
- **fixed** (*int*) – Number of layers to constrain.
- **layer_type** (*'angs' | 'trim' | 'stoich' | 'sym'*) – Method of slab layering to perform. See also: *catkit.gen.surface.SlabGenerator()*

Returns **slabs** – Return a list of enumerated slab structures.

Return type list of Gratoms objects

1.1.4 catkit.db module

class catkit.db.**Atom**(*structure*, *number*, *coordinates*, *constraints*, *magmom*, *charge*)

Bases: sqlalchemy.ext.declarative.api.Base

atom_result

charge

element

element_id

id

magmom

structure

structure_id

x_coordinate

x_fixed

y_coordinate

y_fixed

z_coordinate

z_fixed

class catkit.db.**Atom_Result**(*calculator*, *atom*, *forces=None*)

Bases: sqlalchemy.ext.declarative.api.Base

atom

atoms_id

calculator

calculator_id

id

x_force

y_force

z_force

class catkit.db.**Calculator**(*name*, *xc=None*, *kpoints=None*, *energy_cutoff=None*, *parameters=None*)

Bases: sqlalchemy.ext.declarative.api.Base

atom_result

energy_cutoff

entry

id

name

parameters

structure_result

x_kpoints

xc

y_kpoints

z_kpoints

class catkit.db.**Connect**(*engine='sqlite:///example.db'*)

A class for accessing a temporary SQLite database. This function works as a context manager and should be used as follows:

with Connect() as db: (Perform operation here)

add_entry(*images*, *search_keys=None*)

class catkit.db.**Element**(*i*)
Bases: sqlalchemy.ext.declarative.api.Base

atom

covalent_radii

id

mass

symbol

class catkit.db.**Entry**(*structure*, *calculator*, *trajectory*, *natoms*, *energy*, *fmax*, *smax=None*, *search_keys=None*)
Bases: sqlalchemy.ext.declarative.api.Base

calculator

calculator_id

energy

fmax

id

natoms

search_keys

smax

structure

structure_id

trajectory

class catkit.db.**FingerprintDB**(*db_name='fingerprints.db'*, *verbose=False*)
A class for accessing a temporary SQLite database. This function works as a context manager and should be used as follows:

with FingerprintDB() as fpdb: (Perform operation here)

This syntax will automatically construct the temporary database, or access an existing one. Upon exiting the indentation, the changes to the database will be automatically committed.

create_table()
Creates the database table framework used in SQLite. This includes 3 tables: images, parameters, and fingerprints.

The images table currently stores ase_id information and a unqiue string. This can be adapted in the future to support atoms objects.

The parameters table stores a symbol (10 character maximum) for convenient reference and a description of the parameter.

The fingerprints table holds a unique image and parmeter ID along with a float value for each. The ID pair must be unique.

fingerprint_entry(*ase_id*, *param_id*, *value*)
Enters a fingerprint value to the database for a given ase and parameter id.

Parameters

- **ase_id** (*int*) – The unique id associated with an atoms object in the database.

- **param_id** (*int or str*) – The parameter ID or symbol associated with and entry in the parameters table.
- **value** (*float*) – The value of the parameter for the atoms object.

get_fingerprints(*ase_ids=None*, *params=[]*)
Get the array of values associated with the provided parameters for each ase_id.

Parameters

- **ase_id** (*list*) – The ase-id associated with an atoms object in the database.
- **params** (*list*) – List of symbols or int in parameters table to be selected.

Returns **fingerprint** – An array of values associated with the given parameters (a fingerprint) for each ase_id.

Return type array (n,)

get_parameters(*selection=None*, *display=False*)
Get an array of integer values which correspond to the parameter IDs for a set of provided symbols.

Parameters

- **selection** (*list*) – Symbols in parameters table to be selected. If no selection is made, return all parameters.
- **display** (*bool*) – Print parameter descriptions.

Returns **parameter_ids** – Integer values of selected parameters.

Return type array (n,)

image_entry(*d*, *identity=None*)
Enters a single ase-db image into the fingerprint database. The ase-db ID with identity must be unique. If not, it will be skipped.

This table can be expanded to contain atoms objects in the future.

Parameters

- **d** (*ase-db object*) – Database entry to parse.
- **identity** (*str*) – An identifier of the users choice.

Returns **ase_id** – The ase id collected.

Return type int

parameter_entry(*symbol=None*, *description=None*)
Enters a unique parameter into the database.

Parameters

- **symbol** (*str*) – A unique symbol the entry can be referenced by. If None, the symbol will be the ID of the parameter as a string.
- **description** (*str*) – A description of the parameter.

class catkit.db.**Structure**(*cell*, *pbc*)
Bases: sqlalchemy.ext.declarative.api.Base

atoms

entry

id

pbc

structure_result

x1_cell

x2_cell

x3_cell

y1_cell

y2_cell

y3_cell

z1_cell

z2_cell

z3_cell

class catkit.db.**Structure_Result**(*calculator*, *structure*, *energy*, *stress=None*)

Bases: sqlalchemy.ext.declarative.api.Base

calcualtor_id

calculator

energy

id

structure

structure_id

xx_stress

xy_stress

xz_stress

yy_stress

yz_stress

zz_stress

class catkit.db.**Trajectories**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

entry_id

id

structure_id

1.1.5 catkit.gratoms module

class catkit.gratoms.**Gratoms**(*symbols=None*, *positions=None*, *numbers=None*, *tags=None*, *momenta=None*, *masses=None*, *magmoms=None*, *charges=None*, *scaled_positions=None*, *cell=None*, *pbc=None*, *celldisp=None*, *constraint=None*, *calculator=None*, *info=None*, *edges=None*)

Bases: ase.atoms.Atoms

Graph based atoms object.

An Integrated class for an ASE atoms object with a corresponding Networkx Graph.

adj

connectivity

copy()
Return a copy.

degree

edges

get_chemical_tags(*rank=2*)
Generate a hash descriptive of the chemical formula (rank 0) or include bonding (rank 1).

get_neighbor_symbols(*u*)
Get chemical symbols for neighboring atoms of u.

get_surface_atoms()
Return surface atoms.

get_unsaturated_nodes(*screen=None*)

graph

is_isomorph(*other*)
Check if isomorphic by bond count and atomic number.

nodes

set_surface_atoms(*top*, *bottom=None*)
Assign surface atoms.

1.1.6 catkit.learn module

catkit.learn.**online_learning**(*X, y, samples, factors=[1.0, 1.0], nsteps=40, plot=False*)
A simple utility for performing online learning. The main components required are a regression method and a scoring technique.

Currently, the scoring methodology and regressor are baked in. These need to be made modular.

Minimum 3 samples are required for 3 fold cross validation.

catkit.learn.**optimizer**(*obj_func*, *initial_theta*, *bounds*, *gradient=True*, *minimizer='L-BFGS-B'*, *hopping=0*, ***kwargs*)
Substitute optimizer in scikit-learn Gaussian Process function.

Note 'L-BFGS-B' is equivalent to the standard optimizer used in scikit-learn. This function allows for more direct control over the arguments. https://docs.scipy.org/doc/scipy/reference/optimize.html

Parameters

- **obj_func** (*function*) – scikit-learn objective function.
- **initial_theta** (*array (n,)*) – Hyperparameters to be optimized against.
- **bounds** (*list of tuples (n, 2)*) – Lower and upper bounds for each hyper parameter.
- **gradient** (*bool*) – Include the gradient for the optimization function.
- **minimizer** (*str*) – A scipy minimization method.
- **hopping** (*int*) – Perform a number of basin hopping steps.

Returns

- **theta_opt** (*list (n,)*) – Optimized hyperparameters.
- **func_min** (*float*) – Value of the minimized objective function.

Python Module Index

C

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

X

Y

Z