

Overview

Descriptor based analysis is a powerful tool for understanding the trends across various catalysts. In general, the rate of a reaction over a given catalyst is a function of many parameters - reaction energies, activation barriers, thermodynamic conditions, etc. The high dimensionality of this problem makes it very difficult and expensive to solve completely and even a full solution would not give much insight into the rational design of new catalysts. The descriptor based approach seeks to determine a few "descriptors" upon which the other parameters are dependent. By doing this it is possible to reduce the dimensionality of the problem - preferably to 1 or 2 descriptors - thus greatly reducing computational efforts and simultaneously increasing the understanding of trends in catalysis.

The "mkm" Python module seeks to standardize and automate many of the mathematical routines necessary to move from "descriptor space" to reaction rates. The module is designed to be both flexible and powerful. A "reaction model" can be fully defined by a configuration file, thus no new programming is necessary to change the complexity or assumptions of a model. Furthermore, various steps in the process of moving from descriptors to reaction rates have been abstracted into separate Python classes, making it easy to change the methods used or add new functionality. The downside of this approach is that it makes the code more complex. The purpose of this guide is to explain the general structure of the code, as well as its specific functions and how to begin using it.

Useful Definitions: (Note symbols may not appear in Safari/IE)

- descriptor - variable used to describe reaction kinetics at a high level. Most commonly these are the binding energies of atomic constituents in the adsorbates (e.g. carbon/nitrogen/oxygen adsorption energies) or other intermediates. However, in the general sense other variables such as electronic structure parameters (e.g. d-band center) or thermodynamic parameters (e.g. temperature/pressure) could also be descriptors.
- descriptor space (**D**) - the space spanned by the descriptors. Usually 2^n .
- parameter space (**P**) - the space spanned by the full reaction parameters for the reaction model. Usually 2^n where n is the number of elementary steps (one reaction energy and one reaction barrier per elementary step).
- reaction rates (**r**) - an n-vector of rates corresponding to each of the n elementary reactions.
- descriptor map - a map of some variable (rates, coverages, etc.) as a function of descriptor space.
- reaction model - a set of elementary steps and conditions which define the physics of the kinetic system, along with the mathematical methods and assumptions used to move from "descriptor space" to reaction rates.
- setup file - a file used to define the reaction model
- input file - a file used to store other data which is likely common to many reaction models (e.g. energetics data)

Code Structure:

- The interface to the code is handled through the **ReactionModel** class. This class acts as a messenger class which broadcasts all its attributes to the other classes used in the kinetics module. The class also handles common functions such as printing reactions/adsorbates or comparing /reversing elementary steps. The attributes of **ReactionModel** are automatically synchronized with the parser, scaler, solver, and mapper so it is useful to think of **ReactionModel** as a "toolbox" where all the necessary information and common functions are stored.

The **Parser** class serves to extend the "setup file" by reading in various quantities from an "input file". Technically the use of a parser is optional, but in practice it is extremely helpful for reading in common data such as adsorption energies or vibrational frequencies rather than re-typing them for every reaction model.

The process of creating a "descriptor map" is abstracted into three general processes, which are handled by the following classes within the kinetics module:

Scaler: Projects descriptor space into parameter space: **D** **P**

Solver: Maps parameter space into reaction rates: **P** **r**

Mapper: Moves through descriptor space. This becomes important for practical reasons since it is often necessary to use a solution at one point in descriptor space as an initial guess for a nearby point.

The **ThermoCorrections** class is responsible for applying thermodynamic corrections to the electronic energies which are used as direct inputs. This includes the contributions of entropy/enthalpy and zero-point energy due to temperature, pressure, or other thermodynamic variables.

There are also a variety of analysis classes which allow for automated analysis and visualization of the reaction model. These include the **VectorMap** and **MatrixMap** classes which create plots of outputs as a function of descriptor space. The **VectorMap** class is designed for outputs in the form of vectors (rates, coverages, etc.) while the **MatrixMap** is designed for outputs in the form of matrices (rate control, sensitivity analyses, etc.). The analysis folder also contains **MechanismAnalysis** which creates free energy diagrams, and **ScalingAnalysis** which can give a visual representation of how well the scaler projects descriptor space to parameter space.

Using the code:

Some examples can be found in the "demos" folder, and these should explain the syntax necessary and serve as a good starting point. The currently implemented features are also briefly described below in order to allow a better understanding of the demos and creating original reaction setup files.

Using the kinetics module to conduct a descriptor analysis requires (at least) 2 files: the "setup file" which defines the reaction model, as well as another script to initialize the ReactionModel class and conduct analyses. The "setup file" is generally a static file (i.e. it is not a program) while the submission script will be an actual python program. Setup files typically end in ".mkm" for micro-kinetic model (although this is not required) while submission scripts end in ".py" since they are just python scripts. In addition it is very useful to also have an "input file" which contains the raw data about the energetics of the reaction model. An example of how to create an input file based on a table-like format is given in the [1 - Generating an Input File](#) tutorial.

Each class described in the **Code Structure** section will require some specialized parameters. Some of these parameters are common to all variants of these classes, while others are specific to certain implementations. The possible inputs for the currently implemented variants of each class are listed below. Required attributes are underlined.

- **ReactionModel:**

- **rxn_expressions** - These expressions determine the elementary reaction, and are the most important part of the model. They must be defined unless the elementary_rxns, adsorbate_names, transition_state_names, and gas_names are all explicitly defined since the rxn_expressions are parsed into these 3 attributes. It is much easier to just define rxn_expressions, although it is important to note the syntax. There must be spaces between all elements of each expression (i.e. C*+O* is not okay, but C* + O* is), and species ending with _g are gasses by default. Adsorbed species may end with * or _x where * designates adsorption at the "s" site (by default), while _x designates adsorption at the "x" site (note that "x" may be any letter except "g", and that X* and X_s are equivalent). Transition-states should include a -, and reactions with a transition-state are specified by 'IS <-> TS -> FS' while reactions without a transition-state are defined as 'IS -> FS' (where IS,TS,FS are expressions for the Initial/Transition/Final State). When the model initializes it checks the expressions for mass/site balances, and if it finds that they are not balanced it will raise an exception. [list of strings]. Instead of specifying rxn_expressions the following attributes may instead be defined:
 - elementary_rxns - list version of rxn_expressions. These will be automatically populated if rxn_expressions are defined. [list of lists]
 - adsorbate_names - list of adsorbate names included in the analysis. Automatically populated if rxn_expressions are defined. [list of strings]
 - transition_state_names - list of transition-state names included in the analysis. Automatically populated if rxn_expressions are defined. [list of strings]
 - gas_names - list of gas names included in the analysis. [list of strings]
 - **surface_names** - list of surface names to be included in the analysis. [list of strings]
 - **species_definitions** - This is a dictionary where all species-specific information is stored. The required information will vary depending on the scaler/thermo corrections/solver/mapper used, and the "parser" generally fills in most information. However, there are a few things which generally need to be supplied explicitly:
 - **species_definitions[site][site_names]** (where {site} is each site name in the model) - A list of "site names" which correspond to {site}. If the TableParser (default) is being used then the "site names" must also match the designations in the "site_name" column. For example, if you want the "s" site to correspond to the energetics of an adsorbate at a (211) site, and (211) sites are designated by '211' in the site_name column of the input_file, then this would be specified by: species_definitions['s'] = {'site_names':['211']}. Similarly, if you wanted the 't' site to correspond to 'fcc' or 'bridge' sites then you could specify: species_definitions['t'] = {'site_names':['fcc','bridge']}.
 - **species_definitions[site][total]** (where {site} is each site name in the model) - A number to which the total coverage of {site} must sum. For example, if you wanted to have a total coverage of 1 with 10% 's' sites and 90% 't' sites (with the same site definitions as above) you would specify: species_definitions['s'] = {'site_names':['211'],'total':0.1} and species_definitions['t'] = {'site_names':['fcc','bridge'],'total':0.9}.
 - **species_definitions[gas][pressure]** (where {gas} is each gas name in the model) - The pressure of each gas species in bar. For example, if you wanted a carbon monoxide pressure of 10 bar and hydrogen pressure of 20 bar you would specify: species_definitions['CO_g']['pressure'] = 10 and species_definitions['H2_g']['pressure'] = 20. Note that for some situations you may instead need to specify a 'concentration', 'approach_to_equilibrium', or some other key, but in almost every situation some method for obtaining the gas pressures must be specified for each gas in the model.
 - **temperature** - temperature used for the analysis. May not be defined if ThermodynamicScaler is being used with temperature as a descriptor. [number in Kelvin]
 - **descriptor_names** - names of variables to be used as descriptors. [list of strings]
 - **descriptor_ranges** - Used for mapping through descriptors space. Specify the limits of the descriptor values. Should be a list equal in length to the number of descriptors where each entry is a list of 2 floats (min and max for that descriptor). [list of lists of floats].
 - **resolution** - Used for mapping through descriptor space. Resolution used when discretizing over descriptor_range. [int]
 - **parser** - name of class to use for solver. Defaults to TableParser. [string]
 - **mapper** - name of class to use as a mapper. Defaults to MinResidMapper. [string]
 - **scaler** - name of class to use for scaler. Defaults to GeneralizedLinearScaler. [string]
 - **solver** - name of class to use for solver. Defaults to SteadyStateSolver. [string]
 - **thermodynamics** - name of class to use for thermodynamic corrections. Defaults to ThermoCorrections. [string]
 - **data_file** - file where large outputs will be saved as binary pickle files. Defaults to 'data.pk1' [filepath string]
 - **numerical_representation** - determines how to store numbers as binary. Can be 'mpmath' for multiple precision or 'numpy' for normal floats. Note that 'numpy' rarely works. Defaults to 'mpmath'. [string]
- **Parser:**
- **input_file** - file where input data is stored. File must be in the correct format for the parser used.
See [1 - Generating an Input File](#) for more information.
- **Scaler:**
- **gas_thermo_mode** - Approximation used for obtaining gas-phase free energy corrections. Defaults to ideal_gas. Other possibilities are: shomate_gas (use Shomate equation), zero_point_gas (zero-point corrections only), fixed_entropy_gas (include zero-point and assume entropy is 0.002 eV/K), frozen_gas (no corrections), frozen_zero_point_gas (no zero-point and entropy is 0.002 eV/K). [string]
 - **adsorbate_thermo_mode** - Approximation used for obtaining adsorbate free energy corrections. Defaults to harmonic_adsorbate (use statistical mechanics+vibrational frequencies). Other possibilities are: zero_point_adsorbate (zero-point corrections only), frozen_gas (no corrections). [string]
 - **transition_state_scaling_parameters** - Used if transition-state scaling is used. Many published values are hard-coded, and parameters can often be input directly so it is usually not necessary. Read scalars/_init.py for syntax.
- **Solver:**
- **SteadyStateSolver:**
- **decimal_precision** - number of decimals to explicitly store. Calculation will be slightly slower with larger numbers, but will become completely unstable below some threshold. Defaults to 50. [integer]
 - **tolerance** - all rates must be below this number before the system is considered to be at "steady state". Defaults to 1e-50. [number]
 - **max_rootfinding_iterations** - maximum number of times to iterate the rootfinding algorithm (multi-dimensional Newtons method). Defaults to 50. [integer]
 - **internally_constrain_coverages** - ensure that coverages are greater than 0 and sum to less than the site total within the rootfinding algorithm. Slightly slower, but more stable. Defaults to True. [boolean]
 - **residual_threshold** - the residual must decrease by this proportion in order for the calculation to be considered "converging". Must be less than 1. Defaults to 0.5. [number]
- **Mapper:**
- **MinResidMapper:**

- `search_directions` - list of "directions" to search for existing solutions. Defaults to `[[0,0],[0,1],[1,0],[0,-1],[-1,0],[-1,1],[1,1],[1,-1],[-1,-1]]` which are the nearest points on the orthogonals and diagonals plus the current point. More directions increase the chances of finding a good solution, but slow the mapper down considerably. Note that the current point corresponds to an initial guess coverage provided by the solver (i.e. Boltzmann coverages) and should always be included unless some solutions are already known. [list of lists of integers]
 - `max_bisections` - maximum number of time to bisect descriptor space when moving from one point to the next. Note that this is actually the number of iterations per bisection so that a total of $2^{\text{max_bisections}}$ points could be sampled between two points in descriptor space. Defaults to 3. [integer]
 - `descriptor_decimal_precision` - number of decimals to include when comparing two points in descriptor space. Defaults to 2. [integer]
- **ThermoCorrections:**
 - `thermodynamic_corrections` - corrections to apply. Defaults to `['gas','adsorbate']`. [list of strings]
 - `thermodynamic_variables` - variables/attributes upon which thermo corrections depend. If these variables do not change the corrections will not be updated. Defaults to `['temperatures','gas_pressures']`. [list of strings]
 - `frequency_dict` - used for specifying vibrational frequencies of gasses/adsorbates. Usually populated by the parser. Defaults to `{}`. [dictionary of string:list of numbers in eV]
 - `ideal_gas_params` - parameters used for `ase.thermochemistry.IdealGasThermo`. Defaults to `mkm.data.ideal_gas_params`. [dictionary of string:string/int]
 - `fixed_entropy_dict` - entropies to use in the static entropy approximation. Defaults to `mkm.data.fixed_entropy_dict`. [dictionary of string:float]
 - `atoms_dict` - dictionary of ASE atoms objects to use for `ase.thermochemistry.IdealGasThermo`. Defaults to `ase.structure.molecule(gas_name)`. [dictionary of string:ase.atoms.Atoms]
 - `force_recalculation` - re-calculate thermodynamic corrections even if thermodynamic_variables do not change. Slows the code down considerably, but is useful for sensitivity analyses where thermodynamic variables might be perturbed by very small amounts. Defaults to `False`. [boolean]
 - **Analysis:**
 - **MechanismAnalysis:**
 - `rxn_mechanisms` - dictionary of lists of integers. Each integer corresponds to an elementary step. Elementary steps are indexed in the order that they are input with 1 being the first index. Negative integers are used to designate reverse reactions. [dictionary of string:list of integers]