# ENKIE

**Mattia Gollub**

**Mar 23, 2023**

# CONTENTS

This is the documentation for the ENKIE python package. ENKIE is available on git and via pip. See *getting started* for the installation instructions and a simple example.

ENKIE is a python package for the prediction of kinetic parameter values and uncertainties in metabolic networks. The package uses Bayesian Multilevel Models to predict values and uncertainties from reaction, protein and metabolite identifiers. The predictions are then combined with standard free energy estimates to ensure thermodynamic consistency.

CHAPTER

ONE

# GETTING STARTED

## 1.1 Requirements

In order to install ENKIE you need:

- Windows, Linux or OSX operating system.
- Python 3.8 or newer.
- R installation (tested with R >= 4.2.0).
- ~3GB of free disk space.

## 1.2 Installation

First, install the R *brms* package (tested with version 2.17.6). In a R console, run:

```
install.packages("brms")
```

Once your system satisfies all the requirements, you can install ENKIE through the Python Package Index:

```
pip install enkie
```

*Note*: at the first run, ENKIE will download the models and identifier mappings, this may take a while.

## 1.3 Usage

### 1.3.1 Command Line Interface

Once installed, ENKIE can be used from the command line with the `enkie` command. This approach is most suitable to estimate parameters for SBML models. Running

```
enkie --help
```

will show the available options. A complete example illustrating the usage on the *e_coli_core* model can be found in the git repository .

### 1.3.2 API

For more advanced applications and for finer control over the estimation process, ENKIE can be accessed through a Python API. An simple example can be found in the git repository . See the complete API reference for additional information.

CHAPTER

TWO

# API REFERENCE

This page contains auto-generated API reference documentation[1].

## 2.1 enkie

### 2.1.1 Subpackages

#### enkie.data

##### Subpackages

#### enkie.data.compartment_parameters

#### enkie.dbs

##### Submodules

#### enkie.dbs.kegg

Methods for accessing the KEGG database.

##### Module Contents

enkie.dbs.kegg.**get_ec_to_rxn_mapping**(*ecs: List[str]*) → pandas.DataFrame

Get a mapping from KEGG reaction identifiers to EC numbers for the given ECs from the KEGG database.

**Parameters**
**ecs** (*List[str]*) – The query EC numbers.

**Returns**
DataFrame mapping reactions to ECs.

**Return type**
pd.DataFrame

[1] Created with sphinx-autoapi

### enkie.dbs.metanetx

Methods for accessing the MetaNetX namespace.

#### Module Contents

**class** enkie.dbs.metanetx.**MetaboliteFormat**

Bases: enum.Enum

Format of a metabolite query.

**IDENTIFIER = 0**

**NAME = 1**

**class** enkie.dbs.metanetx.**MnxFile**

Bases: str, enum.Enum

The MetaNetX data files accessible in ENKIE.

**CHEM_XREF = 'chem_xref'**

**CHEM_PROP = 'chem_prop'**

**REAC_XREF = 'reac_xref'**

**REAC_PROP = 'reac_prop'**

**class** enkie.dbs.metanetx.**Metanetx**

Singleton class for mapping metabolite and reaction identifiers using MetaNetX.

**property chem_xref: pandas.DataFrame**

Get the CHEM_XREF MetaNetX table.

**property chem_prop: pandas.DataFrame**

Get the CHEM_PROP MetaNetX table.

**property reac_xref: pandas.DataFrame**

Get the REAC_XREF MetaNetX table.

**property reac_prop: pandas.DataFrame**

Get the REAC_PROP MetaNetX table.

**property rxn_id_to_mnx_id: Dict[str, str]**

Dictionary mapping from reaction identifiers to MetaNetX identifiers.

**property cmp_id_to_mnx_id: Dict[str, str]**

Dictionary mapping from metabolite identifiers to MetaNetX identifiers.

**property name_to_mnx_compound_map: Dict[str, Set[str]]**

A dictionary mapping from (lower case) compound names to metanetx compound identifiers.

**property mnx_id_to_mass: Dict[str, int]**

Dictionary mapping from MetaNetX metabolite identifiers to masses.

**property mnx_id_to_formula_map: Dict[str, str]**

Dictionary mapping from MetaNetX reaction identifiers to reaction formulas.

COLUMNS

MNX_URL_PREFIX = 'https://www.metanetx.org/cgi-bin/mnxget/mnxref/'

REACTIONS_CURATION_FILE = 'data/reaction_mappings_curation.csv'

METABOLITES_CURATION_FILE = 'data/metabolite_mappings_curation.csv'

static get_data_path() → path.Path

Get the path to the folder containing the cached mapping files.

to_mnx_reaction(*query_id: str*, *substrates: Set[str] = None*, *products: Set[str] = None*, *metabolite_format:* MetaboliteFormat = *MetaboliteFormat.IDENTIFIER*, *default: str = None*) → Tuple[str, bool]

Map the given reaction identifier to a MetaNetX identifier, checking whether the mapping preserves directionality.

**Parameters**

- **query_id** (*str*) – The query identifier, in the form <namespace>:<identifier>.
- **substrates** (*Set[str], optional*) – The reaction substrates, by default None
- **products** (*Set[str], optional*) – The reaction products, by default None
- **metabolite_format** (*MetaboliteFormat, optional*) – Specifies the format of the substrate and products, by default MetaboliteFormat.IDENTIFIER
- **default** (*str, optional*) – Value to return if no mapping is found, by default None

**Returns**

The MetaNetX identifier of the reaction, and a flag denoting whether the MetaNetX reaction is defined in the same direction as the input reaction.

**Return type**

Tuple[str, bool]

to_mnx_compound(*query_id: str*, *default: str = None*) → Optional[str]

Map the given compound identifier to a MetaNetX compound.

**Parameters**

- **query_id** (*str*) – The query identifier, in the format <namespace>:<identifier>.
- **default** (*str, optional*) – Value to return if no mapping is found, by default None

**Returns**

The MetaNetX identifier.

**Return type**

Optional[str]

get_compound_mass(*mnx_id: str*) → float

Get the mass of the given compound.

**Parameters**

**mnx_id** (*str*) – The MetaNetX identifier of the query compound.

**Returns**

The mass of the compound, in g/mol.

**Return type**

float

is_forward(*reaction_formula: Optional[str]*, *substrate_ids: Set[str]*, *product_ids: Set[str]*) → Optional[bool]

Detects whether the reaction formula is defined in the forward direction with respect to the given substrates and products.

**Parameters**

- **reaction_formula** (*Optional[str]*) – The MetaNetX reaction formula.
- **substrate_ids** (*Set[str]*) – The identifiers fo teh substrates.
- **product_ids** (*Set[str]*) – The identifiers fo the products.

**Returns**
True if the formula and the substrate/products show the same direction.

**Return type**
Optional[bool]

### enkie.dbs.uniprot

Methods querying protein data from Uniprot.

#### Module Contents

enkie.dbs.uniprot.**FAMILY_LEVELS = ['superfamily', 'family', 'subfamily', 'subsubfamily', 'other_families']**

enkie.dbs.uniprot.**join_protein_ids**(*ids: Iterable[str]*) → str

Join multiple protein identifiers in a single string.

**Parameters**
**ids** (*Iterable[str]*) – The input identifiers.

**Returns**
A string containing the input identifiers in standardized form.

**Return type**
str

enkie.dbs.uniprot.**clean_and_sort_protein_ids**(*ids: str*) → str

Standardize the format of a string containing multiple protein identifiers.

**Parameters**
**ids** (*str*) – The input string.

**Returns**
A string containing the input identifiers in standardized form.

**Return type**
str

enkie.dbs.uniprot.**query_protein_data**(*protein_ids: List[str]*, *columns: List[str]*) → pandas.DataFrame

Query data from Uniprot for the given proteins.

**Parameters**

- **protein_ids** (*List[str]*) – The query Uniprot identifiers.
- **columns** (*List[str]*) – The data columns to return.

**Returns**
The requested protein data.

**Return type**
pd.DataFrame

enkie.dbs.uniprot.**parse_family_string**(*families_string: str*) → Tuple[str, str, str, str, str]

Extract structured protein family information from a Uniprot protein family annotation.

**Parameters**
**families_string** (*str*) – The uniprot family annotation.

**Returns**
The extracted family information, structured as (superfamily, family, subfamily, subsubfamily, other_families).

**Return type**
Tuple[str, str, str, str, str]

**Raises**
**ValueError** – If the input string does not have the expected format.

enkie.dbs.uniprot.**parse_family_df**(*annotations: pandas.DataFrame*) → pandas.DataFrame

Extract structured protein family information from a DataFrame of of Uniprot family annotations.

**Parameters**
**annotations** (*pd.DataFrame*) – The Uniprot family annotations.

**Returns**
The extracted family information, structured as (superfamily, family, subfamily, subsubfamily, other_families).

**Return type**
pd.DataFrame

enkie.dbs.uniprot.**combine_family_names**(*families_df: pandas.DataFrame*, *level: str*) → str

Combine structured protein family information in a single string.

**Parameters**

- **families_df** (*pd.DataFrame*) – The input structured family information.
- **level** (*str*) – The level (one of superfamily, family, subfamily, subsubfamily, other_families) at which information should be combined.

**Returns**
The combined family information.

**Return type**
str

### enkie.estimators

Thermodynamics and kinetics estimation methods and classes.

## Submodules

### enkie.estimators.bmm_kinetic_estimator

Estimator of kinetic parameters based on Bayesian Multilevel Models.

## Module Contents

**class** enkie.estimators.bmm_kinetic_estimator.**BmmKineticEstimator**

Bases: *enkie.estimators.kinetics_estimator_interface.KineticsEstimatorInterface*

An estimator of kinetic parameters based on Bayesian Multilevel Models.

**get_parameters**(*reactions: List[*enkie.miriam_reaction.MiriamReaction*]*, *enzymes: List[*enkie.enzyme.Enzyme*]*, *parameter_types: List[enkie.estimators.kinetics_estimator_interface.KineticParameterType]*, *substrates: List[*enkie.miriam_metabolite.MiriamMetabolite*]*) → Tuple[numpy.ndarray, numpy.ndarray]

Estimate kinetic parameter values for the given reaction-enzyme pairs.

**Parameters**

- **reactions** (*List[MiriamReaction]*) – The reactions to predict parameters for.
- **enzymes** (*List[Enzyme]*) – The enzymes associated with the reactions.
- **parameter_types** (*List[KineticParameterType]*) – The type of the parameters to predict.
- **substrates** (*List[MiriamMetabolite]*) – For KM parameters, the metabolite to predict the KM for. This is ignored for kcat.

**Returns**

The vector of mean parameter ln-values and the covariance of the uncertainty of the estimated parameter ln-values.

**Return type**

Tuple[np.ndarray, np.ndarray]

**predict**(*query_df: pandas.DataFrame*) → Tuple[numpy.ndarray, numpy.ndarray]

Predict kinetic parameters for the given query.

**Parameters**

**query_df** (*pd.DataFrame*) – DataFrame containing the query data.

**Returns**

The mean and covariance of the predicted parameters, in natural log scale.

**Return type**

Tuple[np.ndarray, np.ndarray]

### enkie.estimators.equilibrator_gibbs_estimator

Estimation of Gibbs free energies using eQuilibrator.

## Module Contents

**class** enkie.estimators.equilibrator_gibbs_estimator.**EquilibratorGibbsEstimator**(*rmse_inf: enkie.commons.Q = DEFAULT_RMSE*)

Bases: *enkie.estimators.gibbs_estimator_interface.GibbsEstimatorInterface*

Estimation of Gibbs free energies using *equilibrator_api*.

**Parameters**
**rmse_inf** (*Q, optional*) – Uncertainty to use for unknown groups or compounds.

**property eq_api**

Gets the equilibrator API object.

**property rmse_inf**

Gets the uncertainty used for unknown compounds or groups.

**property incorrect_metabolites: List[str]**

Gets the list of metabolites that are not correctly recognized by equilibrator.

**get_dfg0_prime**(*S: numpy.array*, *metabolites: List[*enkie.miriam_metabolite.MiriamMetabolite*]*, *parameters:* enkie.compartment_parameters.CompartmentParameters) → Tuple[enkie.commons.Q, enkie.commons.Q]

Estimates the standard Gibbs free energies for a reaction network using *equilibrator-api*.

**Parameters**

- **S** (*np.array*) – n-by-m stoichiometric matrix of the reaction network.
- **metabolites** (*List[Metabolite]*) – A m-elements list describing the compounds in the network.
- **compartment_parameters** (CompartmentParameters) – The prior for the physiological parameters of each compartment, such as pH and ionic strength.

**Returns**
A tuple, whose first element is the vector of the mean estimate, and the second is a square root $Q$ of the covariance matrix on the estimation uncertainty $\Sigma$, such that $QQ^\intercal = \Sigma$.

**Return type**
Tuple[Q, Q]

### enkie.estimators.fixed_kinetics_estimator

Mock estimator of kinetic parameters.

## Module Contents

class enkie.estimators.fixed_kinetics_estimator.**FixedKineticsEstimator**(*ln_km_mean: float = np.log(0.0001), ln_km_std: float = 5, ln_kcat_fw_mean: float = np.log(1), ln_kcat_fw_std: float = 5, ln_kcat_bw_mean: float = np.log(1), ln_kcat_bw_std: float = 5*)

Bases: *enkie.estimators.kinetics_estimator_interface.KineticsEstimatorInterface*

Mock estimator of kinetic parameters that returns predefined values.

**Parameters**

- **ln_km_mean** (*float, optional*) – Default value for KM parameters.
- **ln_km_std** (*float, optional*) – Default uncertainty for KM parameters.
- **ln_kcat_fw_mean** (*float, optional*) – Default value for forward kcat parameters.
- **ln_kcat_fw_std** (*float, optional*) – Default uncertainty for forward kcat parameters.
- **ln_kcat_bw_mean** (*float, optional*) – Default value for backward kcat parameters.
- **ln_kcat_bw_std** (*float, optional*) – Default uncertainty for backward kcat parameters.

**get_parameters**(*reactions: List[*enkie.miriam_reaction.MiriamReaction*]*, *enzymes: List[*enkie.enzyme.Enzyme*]*, *parameter_types: List[enkie.estimators.kinetics_estimator_interface.KineticParameterType]*, *substrates: List[*enkie.miriam_metabolite.MiriamMetabolite*]*) → Tuple[numpy.ndarray, numpy.ndarray]

Return default kinetic parameters for teh given reaction-enzyme pairs.

**Parameters**

- **reactions** (*List[MiriamReaction]*) – The reactions to predict parameters for.
- **enzymes** (*List[Enzyme]*) – The enzymes associated with the reactions.
- **parameter_types** (*List[KineticParameterType]*) – The type of the parameters to predict.
- **substrates** (*List[MiriamMetabolite]*) – For KM parameters, the metabolite to predict the KM for. This is ignored for kcat.

**Returns**

The vector of mean parameter ln-values and the covariance of the uncertainty of the estimated parameter ln-values.

**Return type**

Tuple[np.ndarray, np.ndarray]

### enkie.estimators.gibbs_estimator_interface

Interface for an estimator of Gibbs free energies.

#### Module Contents

**class** enkie.estimators.gibbs_estimator_interface.**GibbsEstimatorInterface**

Interface for a class implementing estimation of Gibbs free energies

**abstract get_dfg0_prime**(*S: numpy.array*, *metabolites: List[*enkie.miriam_metabolite.MiriamMetabolite*]*, *parameters:* enkie.compartment_parameters.CompartmentParameters) → Tuple[enkie.commons.Q, enkie.commons.Q]

Estimates the standard Gibbs free energies for a reaction network

**Parameters**

- **S** (*np.array*) – n-by-m stoichiometric matrix of the reaction network.
- **metabolites** (*List[*Metabolite*]*) – A m-elements list describing the compounds in the network.
- **parameters** (CompartmentParameters) – The prior for the physiological parameters of each compartment, such as pH and ionic strength.

**Returns**

A tuple, whose first element is the vector of the mean estimate, and the second is a square root $Q$ of the covariance matrix on the estimation uncertainty $\Sigma$, such that $QQ^\intercal = \Sigma$.

**Return type**

Tuple[Q, Q]

**Raises**

**NotImplementedError** – An metaclass does not implement methods. Please use an implementation of this interface.

### enkie.estimators.kinetic_parameter_types

Definition of types of kinetic parameters.

#### Module Contents

**class** enkie.estimators.kinetic_parameter_types.**KineticParameterType**

Bases: enum.Enum

Type of kinetic parameters supported in ENKIE.

**K_M = 1**

**K_CAT_FORWARD = 2**

**K_CAT_BACKWARD = 3**

### enkie.estimators.kinetics_estimator_interface

Interface for an estimator of kinetic parameters.

#### Module Contents

**class** enkie.estimators.kinetics_estimator_interface.**KineticsEstimatorInterface**

Interface for a class implementing estimation of kinetic parameters

**abstract get_parameters**(*reactions: List[*enkie.miriam_reaction.MiriamReaction*]*, *enzymes: List[*enkie.enzyme.Enzyme*]*, *parameter_types: List[*enkie.estimators.kinetic_parameter_types.KineticParameterType*]*, *substrates: List[*enkie.miriam_metabolite.MiriamMetabolite*]*) → Tuple[numpy.ndarray, numpy.ndarray]

Estimate kinetic parameter values for the given reaction-enzyme pairs.

**Parameters**

- **reactions** (*List[MiriamReaction]*) – The reactions to predict parameters for.
- **enzymes** (*List[Enzyme]*) – The enzymes associated with the reactions.
- **parameter_types** (*List[KineticParameterType]*) – The type of the parameters to predict.
- **substrates** (*List[MiriamMetabolite]*) – For KM parameters, the metabolite to predict the KM for. This is ignored for kcat.

**Returns**

The vector of mean parameter ln-values and the covariance of the uncertainty of the estimated parameter ln-values.

**Return type**

Tuple[np.ndarray, np.ndarray]

### enkie.estimators.parameter_balancer

Estimate kinetic and thermodynamic parameters using parameter balancing.

#### Module Contents

**exception** enkie.estimators.parameter_balancer.**NoEstimateError**

Bases: Exception

Raised when no estimate was returned for a parameter.

**class** enkie.estimators.parameter_balancer.**ParameterBalancer**(*gibbs_estimator: enkie.estimators.gibbs_estimator_interface.GibbsEstimat = None*, *kinetics_estimator: enkie.estimators.kinetics_estimator_interface.KineticsEs = None*, *prior_file: Union[pathlib.Path, str] = None*)

Bases: object

An estimator of kinetic and thermodynamic parameter values based on parameter balancing.

**Parameters**

- **gibbs_estimator** (GibbsEstimatorInterface, *optional*) – The estimator of Gibbs free energies, by default None
- **kinetics_estimator** (KineticsEstimatorInterface, *optional*) – The estimator of kinetic parameters, by default None
- **prior_file** (*Union[Path, str], optional*) – Path to the file defining priors for the Michaelis and velocity constants, by default None

**Raises**

FileNotFoundError – If the specified file does not exist.

**property gibbs_estimator:**
*enkie.estimators.gibbs_estimator_interface.GibbsEstimatorInterface*

The object used to estimate Gibbs free energies.

**property kinetics_estimator:**
*enkie.estimators.kinetics_estimator_interface.KineticsEstimatorInterface*

The object used to estimate kinetic parameters.

**property kv_prior:** *enkie.distributions.LogNormalDistribution*

Prior distribution of velocity constants.

**property km_prior:** *enkie.distributions.LogNormalDistribution*

Prior distribution of affinity constants.

**estimate_parameters**(*reactions: List[*enkie.reaction.Reaction*]*, *rate_laws: List[*enkie.modular_rate_law.ModularRateLaw*]*, *enzymes: List[*enkie.enzyme.Enzyme*]*, *metabolites: List[*enkie.metabolite.Metabolite*]*, *parameters:* enkie.compartment_parameters.CompartmentParameters = *None*) → Tuple[pandas.Series, pandas.DataFrame]

Estimate kinetic and thermodynamic parameter values for the given reactions and metabolites.

**Parameters**

- **reactions** (*List[*Reaction*]*) – The reactions for which reaction energies should be estimated.
- **rate_laws** (*List[*ModularRateLaw*]*) – The rate laws for which kinetic parameter values should be estimated.
- **enzymes** (*List[*Enzyme*]*) – The enzymes associated with the rate laws.
- **metabolites** (*List[*Metabolite*]*) – The metabolites participating to the reactions.
- **parameters** (CompartmentParameters, *optional*) – The physiological parameters of the reaction compartments, by default None

**Returns**

The mean and covariance (representing the uncertainty) of the predicted parameter values.

**Return type**

Tuple[pd.Series, pd.DataFrame]

## enkie.io

### Submodules

## enkie.io.cobra

Methods for parsing metabolites and reactions from cobra models.

### Module Contents

**class** enkie.io.cobra.**EnzymeFactoryInterface**

Interface for a class implementing creation methods for enzyme objects

**property uniprot_fields: Set[str]**

Gets the Uniprot fields required by this class to create Enzyme instances.

**abstract create**(*ec: str*, *uniprot_ids: List[str]*, *gene_ids: List[str]*, *uniprot_data: pandas.DataFrame*) → *enkie.enzyme.Enzyme*

Create and Enzyme instance.

**Parameters**

- **ec** (*str*) – EC number of the enzyme.
- **uniprot_ids** (*List[str]*) – The Uniprot identifiers of the proteins included in the enzyme.
- **gene_ids** (*List[str]*) – The identifiers of the genes included in the enzyme.
- **uniprot_data** (*pd.DataFrame*) – The data retrieved from Uniprot for this enzyme.

**Returns**
The created instance.

**Return type**
*Enzyme*

**Raises**
**NotImplementedError** – If the method is not implemented in the child class.

**class** enkie.io.cobra.**EnzymeFactory**

Bases: *EnzymeFactoryInterface*

Factory class for the construction of Enzyme objects.

**create**(*ec: str*, *uniprot_ids: List[str]*, *gene_ids: List[str]*, *_: pandas.DataFrame*) → *enkie.enzyme.Enzyme*

Create and Enzyme instance.

**Parameters**

- **ec** (*str*) – EC number of the enzyme.
- **uniprot_ids** (*List[str]*) – The Uniprot identifiers of the proteins included in the enzyme.
- **gene_ids** (*List[str]*) – The identifiers of the genes included in the enzyme.
- **uniprot_data** (*pd.DataFrame*) – The data retrieved from Uniprot for this enzyme.

**Returns**
The created instance.

**Return type**
*Enzyme*

enkie.io.cobra.**parse_enzymes**(*model: cobra.Model*, *reaction_ids: List[str]*, *spontaneous_genes: Iterable = None*, *enzyme_factory: EnzymeFactoryInterface = None*) → OrderedDict[str, List[*Enzyme*]]

Parse enzyme information from the given cobra model.

**Parameters**

- **model** (*cobra.Model*) – The target model.
- **reaction_ids** (*List[str]*) – Identifiers of the reactions for which enzyme objects should be constructed.
- **spontaneous_genes** (*Iterable, optional*) – Identifiers of pseudo-genes that represent spontaneous (non enzyme-associated) reactions, by default None
- **enzyme_factory** (*EnzymeFactoryInterface, optional*) – Factory object for the construction of enzyme objects, by default None

**Returns**
The enzymes of each reaction.

**Return type**
OrderedDict[str, List[*Enzyme*]]

enkie.io.cobra.**get_needed_metabolite_ids**(*model: cobra.Model*, *reaction_ids: List[str]*) → List[str]

Get the identifiers of the metabolites participating to the given reactions.

**Parameters**

- **model** (*cobra.Model*) – The target model.
- **reaction_ids** (*List[str]*) – The identifiers of the target reactions.

**Returns**
The identifiers of the metabolites participating to the specified reactions.

**Return type**
List[str]

enkie.io.cobra.**parse_metabolite_id**(*metabolite: cobra.Metabolite*, *metabolites_namespace: str = None*) → str

Get the identifier (including namespace) of a metabolite.

**Parameters**

- **metabolite** (*cobra.Metabolite*) – The target metabolite.
- **metabolites_namespace** (*str, optional*) – The namespace to read the identifier from, by default None.

**Returns**
The metabolite identifier..

**Return type**
str

**Raises**
**ValueError** – If no annotation was found for the given namespace.

enkie.io.cobra.**parse_metabolites**(*model: cobra.Model*, *metabolite_ids: List[str]*, *metabolites_namespace: str = None*) → OrderedDict[str, *Metabolite*]

Parse the given metabolites from a model.

**Parameters**

- **model** (*cobra.Model*) – The target model.
- **metabolite_ids** (*List[str]*) – The identifier of the metabolites that should be parsed.
- **metabolites_namespace** (*str, optional*) – The namespace to read identifiers from, by default None.

**Returns**
Mapping from query IDs to the parsed metabolites.

**Return type**
OrderedDict[str, *Metabolite*]

enkie.io.cobra.**parse_reactions**(*model: cobra.Model*, *metabolites: OrderedDict[str,* Metabolite*]*, *reaction_ids: List[str]*, *reactions_namespace: str = None*) → OrderedDict[str, *Reaction*]

Parse the given reaction from a model.

**Parameters**

- **model** (*cobra.Model*) – The target model.
- **metabolites** (*OrderedDict[str, Metabolite]*) – The metabolites participating to the reactions.
- **reaction_ids** (*List[str]*) – The identifiers of the reaction to be parsed.
- **reactions_namespace** (*str, optional*) – The namespace from which reaction identifiers should be read, by default None

**Returns**
Mapping from input IDs to the parsed reactions.

**Return type**
OrderedDict[str, *Reaction*]

**Raises**
**ValueError** – If a reaction has not identifier annotation for the specified namespace.

enkie.io.cobra.**make_default_rate_laws**(*reactions: Dict[str,* enkie.reaction.Reaction*]*, *reaction_enzymes: Dict[str, List[*enkie.enzyme.Enzyme*]]*, *rate_law_type:* enkie.modular_rate_law.RateLawDenominator *= DEFAULT_RATE_LAW_TYPE*) → Tuple[List[*enkie.modular_rate_law.ModularRateLaw*], List[*enkie.enzyme.Enzyme*]]

Create default rate laws for the given enzymes.

**Parameters**

- **reactions** (*Dict[str, Reaction]*) – The reactions catalyzed by the enzymes.
- **reaction_enzymes** (*Dict[str, List[Enzyme]]*) – Mapping from reaction identifiers to enzymes for which rate laws should be created.
- **rate_law_type** (*RateLawDenominator, optional*) – The rate law type to use, by default DEFAULT_RATE_LAW_TYPE

**Returns**
The created rate laws and the corresponding enzymes.

**Return type**
Tuple[List[*ModularRateLaw*], List[*Enzyme*]]

### 2.1.2 Submodules

#### enkie.commons

Common definitions.

#### Module Contents

enkie.commons.**Q**
Type used for describing quantities.

#### enkie.compartment_parameters

Parameters for the compartments of a biochemical network.

#### Module Contents

enkie.compartment_parameters.**COMPARTMENT_ANY = '__any__'**
Pseudo-identifier representing any compartment.

**class** enkie.compartment_parameters.**CompartmentParameters**(*compartment_pH: Dict[str, enkie.constants.Q] = None*, *compartment_pMg: Dict[str, enkie.constants.Q] = None*, *compartment_I: Dict[str, enkie.constants.Q] = None*, *compartment_phi: Dict[str, enkie.constants.Q] = None*, *T: enkie.constants.Q = DEFAULT_T*)
Parameters for the compartments of a metabolic network.

**Parameters**

- **compartment_pH** (*Dict[str, Q], optional*) – Mapping from compartment identifiers to the pH of the compartment.
- **compartment_pMg** (*Dict[str, Q], optional*) – Mapping from compartment identifiers to the pMg of the compartment.
- **compartment_I** (*Dict[str, Q], optional*) – Mapping from compartment identifiers to the ionic strength of the compartment.
- **compartment_phi** (*Dict[str, Q], optional*) – Mapping from compartment identifiers to the electrostatic potential of the compartment.
- **T** (*Q, optional*) – Temperature of the system (temperature must be the same for all compartments).

**pH**(*compartment: str*) → enkie.constants.Q

Gets the pH of a compartment.

**pMg**(*compartment: str*) → enkie.constants.Q

Gets the pMg of a compartment.

**I**(*compartment: str*) → enkie.constants.Q

Gets the ionic strength of a compartment.

**phi**(*compartment: str*) → enkie.constants.Q

Gets the electrostatic potential of a compartment.

**T**() → enkie.constants.Q

Gets the temperature of the system.

**static load**(*params_file: Union[pathlib.Path, str]*) → *CompartmentParameters*

Loads the compartment parameters from a .csv file.

**Parameters**

**params_file** (*Union[Path, str]*) – Path to the file containing the parameter values or name of a builtin parameter set (any file present in data/compartment_parameters/, e.g. 'e_coli' or 'human').

**Returns**

New instance of this class, containing the parameters loaded from the file.

**Return type**

*CompartmentParameters*

## enkie.constants

### Module Contents

enkie.constants.**R**

The gas constant.

enkie.constants.**F**

The Faraday constant.

enkie.constants.**LOG10**

The natural logarithm of 10.

enkie.constants.**DEFAULT_I**

Default ionic strength of a compartment.

enkie.constants.**DEFAULT_PH**

Default pH of a compartment.

enkie.constants.**DEFAULT_PMG**

Default pMg of a compartment.

enkie.constants.**DEFAULT_PHI**

Default electrostatic potential of a compartment.

enkie.constants.**DEFAULT_T**

Default temperature of the system.

enkie.constants.**DEFAULT_RMSE**
Default uncertainty to use for unknown compounds and chemical groups.

enkie.constants.**DEFAULT_SPONTANEOUS_GENES**
Default list of genes that denote spontaneous reactions.

### enkie.distributions

Descriptions of probability distributions.

#### Module Contents

**class** enkie.distributions.**UniformDistribution**(*lb: float = 0.0*, *ub: float = 1.0*)
Uniform distribution in a given interval.

**Parameters**

- **lb** (*float, optional*) – Lower bound of the interval, by default 0.0
- **ub** (*float, optional*) – Upper bound of the interval, by default 1.0

**property lb: float**
Gets the lower bound of the distribution.

**property ub: float**
Gets the upper bound of the distribution.

**copy**() → *UniformDistribution*
Creates a copy of this object.

**Returns**
Copy of this object.

**Return type**
*UniformDistribution*

**class** enkie.distributions.**NormalDistribution**(*mean: float = 0.0*, *std: float = 1.0*)
Normal distribution with given mean and standard deviation.

**Parameters**

- **mean** (*float, optional*) – Mean of the distribution, by default 0.0
- **std** (*float, optional*) – Standard deviation, by default 1.0

**property mean: float**
Gets the mean of the distribution.

**property std: float**
Gets the standard deviation of the distribution.

**copy**()
Creates a copy of this object.

**Returns**
Copy of this object.

**Return type**
*NormalDistribution*

class enkie.distributions.**LogUniformDistribution**(*lb: float = 0.0*, *ub: float = 1.0*)

Log-uniform distribution in a given interval.

**Parameters**

- **lb** (*float, optional*) – Lower bound of the interval, by default 0.0
- **ub** (*float, optional*) – Upper bound of the interval, by default 1.0

property **lb: float**

Gets the lower bound of the distribution.

property **ub: float**

Gets the upper bound of the distribution.

**copy**()

Creates a copy of this object.

**Returns**
Copy of this object.

**Return type**
*LogUniformDistribution*

class enkie.distributions.**LogNormalDistribution**(*log_mean: float = 0.0*, *log_std: float = 1.0*)

Log-normal distribution with given mean and standard deviation.

**Parameters**

- **log_mean** (*float, optional*) – Natural logarithm of the mean of the distribution, by default 0.0
- **log_std** (*float, optional*) – Standard deviation of the distribution on the log scale, by default 1.0

property **log_mean: float**

Gets the log-mean of the distribution.

property **log_std: float**

Gets the log-standard deviation of the distribution.

**copy**()

Creates a copy of this object.

**Returns**
Copy of this object.

**Return type**
*LogNormalDistribution*

enkie.distributions.**distribution_from_string**(*parameters_string: str*) → Any

Parses a distribution from a string. The format of the string is <distribution>|<data1>|<...>, where distribution is the type of the distribution and the remaining elements the arguments of the constructor of the distribution. For example, "Uniform|1.0|2.0".

**Parameters**
**parameters_string** (*str*) – String describing the distribution.

**Returns**
The constructed distribution object.

**Return type**
Any

**Raises**
Exception – If the type of the distribution is not recognized.

enkie.distributions.**distribution_to_string**(*distribution: Any*) → str

Encodes a distribution in a string. The format of the string is <distribution>|<data1>|<...>, where distribution is the type of the distribution and the remaining elements the arguments of the constructor of the distribution. For example, "Uniform|1.0|2.0".

**Parameters**
**distribution** (*Any*) – Distribution object to be converted.

**Returns**
String describing the distribution.

**Return type**
str

**Raises**
Exception – If the specified distribution is not supported by this method.

### enkie.enzyme

Basic description of an enzyme

#### Module Contents

**class** enkie.enzyme.**Enzyme**(*ec: str*, *uniprot_acs: List[str]*, *gene_ids: List[str]*)

Basic description of an enzyme.

**Parameters**

- **ec** (*str*) – The EC number of the reaction catalyzed by the enzyme.
- **uniprot_acs** (*List[str]*) – The Uniprot identifiers of the proteins included in the enzyme.
- **gene_ids** (*List[str]*) – The identifiers of the genes encoding the proteins of the enzyme.

**property ec: str**

The EC number of the enzyme.

**property uniprot_acs: List[str]**

Uniprot accession identifiers for the proteins composing in the enzyme.

**property gene_ids: List[str]**

Gene identifiers for the proteins composing in the enzyme.

### enkie.metabolite

Basic description of a metabolite which is part of a metabolic model.

### Module Contents

class enkie.metabolite.**Metabolite**(*mid: str*, *miriam_id: str*, *compartment: str = 'c'*, *nH: int = 0*, *z: int = 0*)

Bases: *enkie.miriam_metabolite.MiriamMetabolite*

A metabolite in a metabolic model.

**Parameters**

- **mid** (*str*) – Unique identifier of the metabolite as defined in the model.
- **miriam_id** (*str*) – Identifier of the metabolite as defined on identifiers.org. Such ID has the format "<prefix>:<id>", such as "bigg.metabolite:g6p" or "kegg.compound:C00009". Ideally, only identifiers supported by MetaNetX should be used. Other identifiers will be treated as fictional.
- **compartment** (*str, optional*) – The identifier of the compartment, by default 'c'.
- **nH** (*int, optional*) – Number of hydrogen atoms in the metabolite, by default 0.
- **z** (*int, optional*) – Charge of the metabolite, by default 0.

property **id**: str

Gets the unique identifier of the metabolite.

## enkie.miriam_metabolite

Basic description of a metabolite.

### Module Contents

class enkie.miriam_metabolite.**MiriamMetabolite**(*miriam_id: str*, *compartment: str = 'c'*, *nH: int = 0*, *z: int = 0*)

Describes the properties of a metabolite relevant for the estimation of Gibbs free energies and kinetic parameters.

**Parameters**

- **miriam_id** (*str*) – Identifier of the metabolite as defined on identifiers.org. Such ID has the format "<prefix>:<id>", such as "bigg.metabolite:g6p" or "kegg.compound:C00009". Ideally, only identifiers supported by MetaNetX should be used. Other identifiers will be treated as fictional.
- **compartment** (*str, optional*) – The identifier of the compartment, by default 'c'.
- **nH** (*int, optional*) – Number of hydrogen atoms in the metabolite, by default 0.
- **z** (*int, optional*) – Charge of the metabolite, by default 0.

property **miriam_id**: str

Gets the MIRIAM identifier of the metabolite.

property **metanetx_id**: str

Gets the MetaNetX identifier of the metabolite or None if the metabolite does not exist in MetaNetX.

property **compartment**: str

Gets the compartment ID of the metabolite.

**property nH: int**

Gets the number of hydrogen atoms in the metabolite.

**property z: int**

Gets the charge of the metabolite.

**UNKNOWN_ID = ''**

### enkie.miriam_reaction

Basic description of a reaction.

#### Module Contents

**class** enkie.miriam_reaction.**MiriamReaction**(*miriam_id: str*, *metabolites: List[enkie.miriam_metabolite.MiriamMetabolite]*, *stoichiometry: numpy.ndarray*)

Bases: object

Describes identity and stoichiometry of a reaction.

**Parameters**

- **miriam_id** (*str*) – Identifier of the reaction as defined on identifiers.org. Such ID has the format "<prefix>:<id>", such as "bigg.reaction:PGI" or "kegg.reaction:R00004". Ideally, only identifiers supported by MetaNetX should be used. Other identifiers will be treated as fictional.
- **metabolites** (*List[MiriamMetabolite]*) – The metabolites participating to the reaction.
- **stoichiometry** (*np.ndarray*) – The stoichiometry of the participating metabolites.

**property miriam_id: str**

The MIRIAM identifier of the reaction.

**property metabolites: List[*enkie.miriam_metabolite.MiriamMetabolite*]**

The metabolites involved in the reaction.

**property S: numpy.ndarray**

The stoichiometric vector of the reaction, defined over the metabolites of the rate law.

**property substrates: List[*enkie.miriam_metabolite.MiriamMetabolite*]**

The reaction substrates.

**property products: List[*enkie.miriam_metabolite.MiriamMetabolite*]**

The reaction products.

**property num_metabolites: int**

The number of metabolites (substrates and products) in this reaction.

### enkie.modular_rate_law

Description of modular rate laws.

## Module Contents

**class** enkie.modular_rate_law.**RateLawDenominator**

Bases: enum.Enum

The type of a modular rate law.

**COMMON = 1**

**DIRECT_BINDING = 2**

**SIMULTANEOUS_BINDING = 3**

**POWER_LAW = 4**

**FORCE_DEPENDENT = 5**

enkie.modular_rate_law.**DEFAULT_RATE_LAW_TYPE**

**class** enkie.modular_rate_law.**ModularRateLaw**(*rid: str*, *reaction:* enkie.reaction.Reaction, *denominator:* RateLawDenominator = *DEFAULT_RATE_LAW_TYPE*, *cooperativity: float = 1*)

Bases: object

A reaction following modular rate law kinetics.

**Parameters**

- **rid** (*str*) – Unique identifier of the ate law.
- **reaction** (Reaction) – The reaction whose kinetics are described by this rate law.
- **denominator** (RateLawDenominator, *optional*) – The denominator of the rate law.
- **cooperativity** (*float*, *optional*) – The cooperativity of the reaction.

**property id: str**

The identifier of the rate law.

**property reaction: *enkie.reaction.Reaction***

The reaction associated to the rate law.

**property num_independent_params: int**

The number of independent parameters in the rate law.

**property metabolites_kin: List[*enkie.metabolite.Metabolite*]**

The reaction metabolites participating to the reaction that have an effect on kinetics.

**property metabolite_ids_kin: List[str]**

The identifiers of the metabolites participating to the reaction that have an effect on kinetics.

**property num_metabolites_kin: int**

The number of metabolites (substrates or products) in this reaction that have an effect on the reaction kinetics.

**NO_KM_MNX_IDS**

MetaNetX identifiers of metabolites that are assumed not to affect the kinetics of the reaction.

**get_dependence**(*parameter:* enkie.estimators.kinetic_parameter_types.KineticParameterType, *temperature:* *enkie.commons.Q*, *metabolite:* enkie.metabolite.Metabolite = *None*) → numpy.ndarray

Gets the dependence vector between the independent parameters and the selected parameter of the rate law.

**Parameters**

- **parameter** (KineticParameterType) – The type of parameter to get the dependence vector for.
- **temperature** (*Q*) – The temperature at which the dependence vector is evaluated.
- **metabolite** (Metabolite, *optional*) – The metabolite a KM parameter refers to. Only meaningful for KM parameters.

**Returns**

The dependence vector. The format is [DfG'°, ln kV, ln kMs].

**Return type**

np.ndarray

## enkie.parameter_space

A class representing parameter values and their uncertainty.

### Module Contents

**class** enkie.parameter_space.**ParameterSpace**(*reactions: List[*enkie.reaction.Reaction*]*, *rate_laws:* *List[*enkie.modular_rate_law.ModularRateLaw*]*, *enzymes:* *List[*enkie.enzyme.Enzyme*]*, *metabolites:* *List[*enkie.metabolite.Metabolite*]*, *parameters:* enkie.compartment_parameters.CompartmentParameters = *None*, *estimator:* enkie.estimators.parameter_balancer.ParameterBalancer = *None*)

A class representing parameter values and their uncertainty.

**Parameters**

- **reactions** (*List[*Reaction*]*) – The reactions included in the parameter space.
- **rate_laws** (*List[*ModularRateLaw*]*) – The rate laws describing the reaction kinetics.
- **enzymes** (*List[*Enzyme*]*) – The enzymes associated with the rate laws.
- **metabolites** (*List[*Metabolite*]*) – The metabolites included in the parameter space.
- **parameters** (CompartmentParameters, *optional*) – The physiological parameters of the reaction compartments, by default None
- **estimator** (ParameterBalancer, *optional*) – The object used for estimating kinetic and thermodynamic parameters, by default None

**property core_mean: pandas.Series**

The mean of the core variables (standard formation energies, log velocities and log affinities).

**property core_cov: pandas.DataFrame**

The covariance of the core variables (standard formation energies, log velocities and log affinities).

**property core_to_all**

The linear transform from the core parameters to all parameters.

**property mean: pandas.Series**

The mean of all parameter values (standard formation and reaction energies, log velocities, log catalytic rates and log affinities).

**property cov: pandas.DataFrame**

The covariance of all parameter values (standard formation and reaction energies, log velocities, log catalytic rates and log affinities).

**property metadata: pandas.DataFrame**

The association between rate law identifiers and genes.

**sample**(*num_samples: int*, *parameters: Union[List[str], List[Tuple[str, str]]] = None*) → pandas.DataFrame

Draw parameter samples from their estimated distribution.

**Parameters**

- **num_samples** (*int*) – Number of parameter samples to draw.
- **parameters** (*Union[List[str], List[Tuple[str, str]]], optional*) – The parameters to draw samples for, by default None (all parameters)

**Returns**
The parameter samples.

**Return type**
pd.DataFrame

## enkie.reaction

Basic description of a reaction which is part of a metabolic model.

### Module Contents

**class** enkie.reaction.**Reaction**(*rid: str*, *miriam_id: str*, *metabolites: List[*enkie.metabolite.Metabolite*]*, *stoichiometry: numpy.ndarray*)

Bases: *enkie.miriam_reaction.MiriamReaction*

A reaction in a metabolic model.

**Parameters**

- **rid** (*str*) – Unique identifier of the reaction as defined in the model.
- **miriam_id** (*str*) – Identifier of the reaction as defined on identifiers.org. Such ID has the format "<prefix>:<id>", such as "bigg.reaction:PGI" or "kegg.reaction:R00004". Ideally, only identifiers supported by MetaNetX should be used. Other identifiers will be treated as fictional.
- **metabolites** (*List[MiriamMetabolite]*) – The metabolites participating to the reaction.
- **stoichiometry** (*np.ndarray*) – The stoichiometry of the participating metabolites.

**property id: str**

The unique identifier of the reaction.

**property metabolite_ids: List[str]**

The identifiers of the metabolites participating to the reaction.

### enkie.singleton_meta

Metaclass for singleton classes.

#### Module Contents

**class** enkie.singleton_meta.**SingletonMeta**(*name*, *bases*, *dic*)

Bases: type

Metaclass for singleton classes. Credits: https://www.pythonprogramming.in/singleton-class-using-metaclass-in-python.html

### enkie.storage

Utility methods for managing storage of cached data.

#### Module Contents

enkie.storage.**DEFAULT_KCAT_MODEL**

enkie.storage.**DEFAULT_KM_MODEL**

enkie.storage.**get_data_path**() → pathlib.Path

Gets the path usage for storing cached data.

**Returns**
Tha cache path.

**Return type**
Path

enkie.storage.**clear_enkie_cache**() → None

Clears the cache of the enkie package. This includes cache MetaNetX mapping files and cached Uniprot requests.

enkie.storage.**get_cached_filepath**(*settings: equilibrator_cache.zenodo.ZenodoSettings*) → pathlib.Path

Get data from a file stored in Zenodo (or from cache, if available). Based on code by Elad Noor and Moritz Beber in equilibrator_cache,: https://gitlab.com/equilibrator/equilibrator-cache/-/blob/2249b3f334ebe8eed9b62475644bf7a2e385cde1/src/equilibrator_cache/zenodo.py

**Parameters**
**settings** (*ZenodoSettings*) – Configuration for the interaction with Zenodo.org.

**Returns**
The path to the locally cached file.

**Return type**
Path

### enkie.utils

General utility methods.

## Module Contents

enkie.utils.**get_internal_reaction_ids**(*model: cobra.Model*) → List[str]

Get the identifier of the internal reactions in the model, i.e. reactions that are not exchanges, biomass, demands or sinks.

**Parameters**
**model** (*cobra.Model*) – The target model.

**Returns**
The identifiers of the internal reactions.

**Return type**
List[str]

enkie.utils.**get_path**(*path: Union[pathlib.Path, str]*) → pathlib.Path

Gets a `Path` object from different representations.

**Parameters**
**path** (*Union[Path, str]*) – A `Path` object or a string describing the path.

**Returns**
A `Path` object.

**Return type**
Path

**Raises**
**ValueError** – If the type of the input is not supported.

enkie.utils.**make_stoichiometric_matrix**(*reactions: List[*enkie.miriam_reaction.MiriamReaction*]*, *metabolites: List[*enkie.miriam_metabolite.MiriamMetabolite*]*) → numpy.ndarray

Make a stoichiometric matrix for a given network.

**Parameters**

- **reactions** (*List[MiriamReaction]*) – The reactions in the network.
- **metabolites** (*List[MiriamMetabolite]*) – The metabolites in the network.

**Returns**
The stoichiometric matrix of the network.

**Return type**
np.ndarray

enkie.utils.**qvector**(*elements: List[enkie.commons.Q]*) → enkie.commons.Q

Converts a list of quantities to a quantity vector.

**Parameters**
**elements** (*List[Q]*) – List of quantities.

**Returns**
Quantity vector.

**Return type**
Q

enkie.utils.**qrvector**(*elements: List[enkie.commons.Q]*) → enkie.commons.Q

Converts a list of quantities to a quantity row vector.

**Parameters**
**elements** (*List[Q]*) – List of quantities.

**Returns**
Quantity row vector.

**Return type**
Q

enkie.utils.**to_reactions_idxs**(*reactions: Union[List[int], List[str], cobra.DictList, List[cobra.Reaction]]*, *model: cobra.Model*) → List[int]

Utility function to obtain a list of reaction indices from different representations.

**Parameters**

- **reactions** (*Union[List[int], List[str], cobra.DictList, List[cobra.Reaction]]*) – Input list of reactions. Reactions can be defined through their index in the model, their identifiers, or with the reactions themselves.
- **model** (*cobra.Model*) – The model in which the reactions are defined.

**Returns**
List of reaction indices.

**Return type**
List[int]

**Raises**
**ValueError** – If the list is not of one of the expected formats.

## 2.2 cli

Command Line Interface for ENKIE.

### 2.2.1 Module Contents

cli.**main**(*sbml_file*, *prefix*, *metabolites_namespace*, *reactions_namespace*, *compartment_params_file*, *clear_cache*)

Estimates kinetic and thermodynamic parameters from the SBML_FILE model and saves the mean and covariance of the prediction in <PREFIX>_mean.csv and <PREFIX>_cov.csv. Additionally saves the association between per-enzyme reactions and genes in <PREFIX>_genes.csv.

# PYTHON MODULE INDEX

## c



## e

# INDEX

## F



## G



## I

## N



## P



## Q



## R



## S

## T



## U



## Z