pyglotaran Documentation

Release 0.3.0

Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum

Feb 23, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Pyglotaran is a python library for global analysis of time-resolved spectroscopy data. It is designed to provide a state of the art modeling toolbox to researchers, in a user-friendly manner.

Its features are:

- user-friendly modeling with a custom YAML (`*.yml`) based modeling language
- parameter optimization using variable projection and non-negative least-squares algorithms
- easy to extend modeling framework
- battle-hardened model and algorithms for fluorescence dynamics
- build upon and fully integrated in the standard Python science stack (NumPy, SciPy, Jupyter)

1.1 A Note To Glotaran Users

Although closely related and developed in the same lab, pyglotaran is not a replacement for Glotaran - A GUI For TIMP. Pyglotaran only aims to provide the modeling and optimization framework and algorithms. It is of course possible to develop a new GUI which leverages the power of pyglotaran (contributions welcome).

The current 'user-interface' for pyglotaran is Jupyter Notebook. It is designed to seamlessly integrate in this environment and be compatible with all major visualization and data analysis tools in the scientific python environment.

If you are a non-technical user, you should give these tools a try, there are numerous tutorials how to use them. You don't need to really learn to program. If you can use e.g. Matlab or Mathematica, you can use Jupyter and Python.

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Python 3.6 or later

2.1.1 Windows

The easiest way of getting Python (and some basic tools to work with it) in Windows is to use Anaconda, which provides python.

You will need a terminal for the installation. One is provided by *Anaconda* and is called *Anaconda Console*. You can find it in the start menu.

Note: If you use a Windows Shell like cmd.exe or PowerShell, you might have to prefix '$PATH_TO_ANACONDA/' to all commands (e.g. *C:/Anaconda/pip.exe* instead of *pip*)

2.2 Stable release

Warning: pyglotaran is early development, so for the moment stable releases are sparse and outdated. We try to keep the master code stable, so please install from source for now.

This is the preferred method to install pyglotaran, as it will always install the most recent stable release.

To install pyglotaran, run this command in your terminal:

```
$ pip install pyglotaran
```

If you don't have pip installed, this Python installation guide can guide you through the process.

If you want to install it via conda, you can run the following command:

```
$ conda install -c conda-forge pyglotaran
```

2.3 From sources

First you have to install or update some dependencies.

Within a terminal:

```
$ pip install -U numpy scipy Cython
```

Alternatively, for Anaconda users:

```
$ conda install numpy scipy Cython
```

Afterwards you can simply use pip to install it directly from Github.

```
$ pip install git+https://github.com/glotaran/pyglotaran.git
```

For updating pyglotaran, just re-run the command above.

If you prefer to manually download the source files, you can find them on Github. Alternatively you can clone them with git (preferred):

```
$ git clone https://github.com/glotaran/pyglotaran.git
```

Within a terminal, navigate to directory where you have unpacked or cloned the code and enter

```
$ pip install -e .
```

For updating, simply download and unpack the newest version (or run `$ git pull` in pyglotaran directory if you used git) and and re-run the command above.

CHAPTER

THREE

QUICKSTART/CHEAT-SHEET

Warning: pyglotaran is in very early stage of development. You should not use it for actual science at the moment.

To start using pyglotaran in your project, you have to import it first. In addition we need to import some extra components for later use.

```
In [1]: import glotaran as gta

In [2]: from glotaran.analysis.optimize import optimize

In [3]: from glotaran.analysis.scheme import Scheme
```

Let us get some data to analyze:

```
In [4]: from glotaran.examples.sequential import dataset

In [5]: dataset
Out[5]:
<xarray.Dataset>
Dimensions:   (spectral: 72, time: 2100)
Coordinates:
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 0.004229 0.03016 0.004733 ... 1.714 1.543
```

Like all data in `pyglotaran`, the dataset is a `xarray.Dataset`. You can find more information about the `xarray` library the xarray hompage.

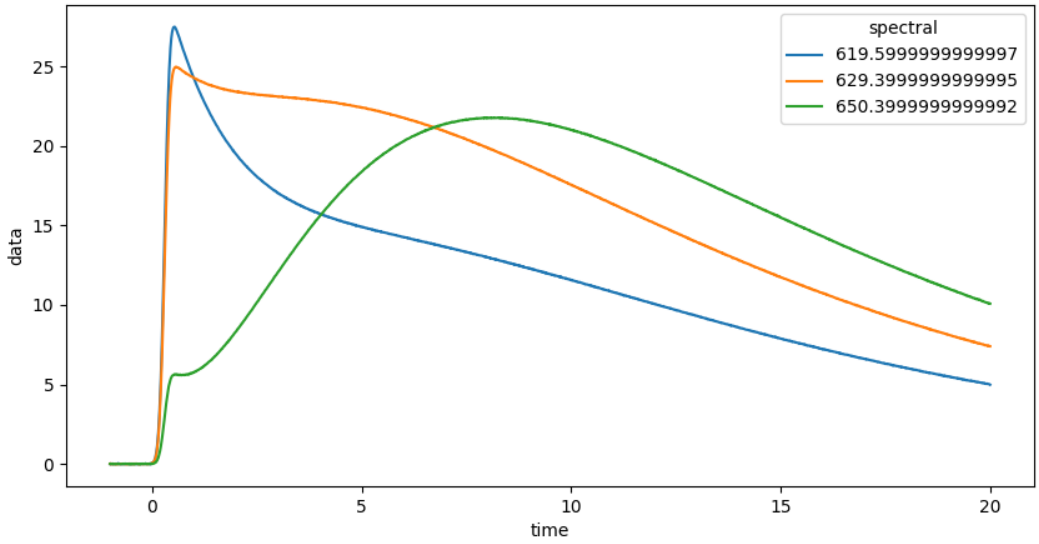
The loaded dataset is a simulated sequential model.

To plot our data, we must first import matplotlib.

```
In [6]: import matplotlib.pyplot as plt
```

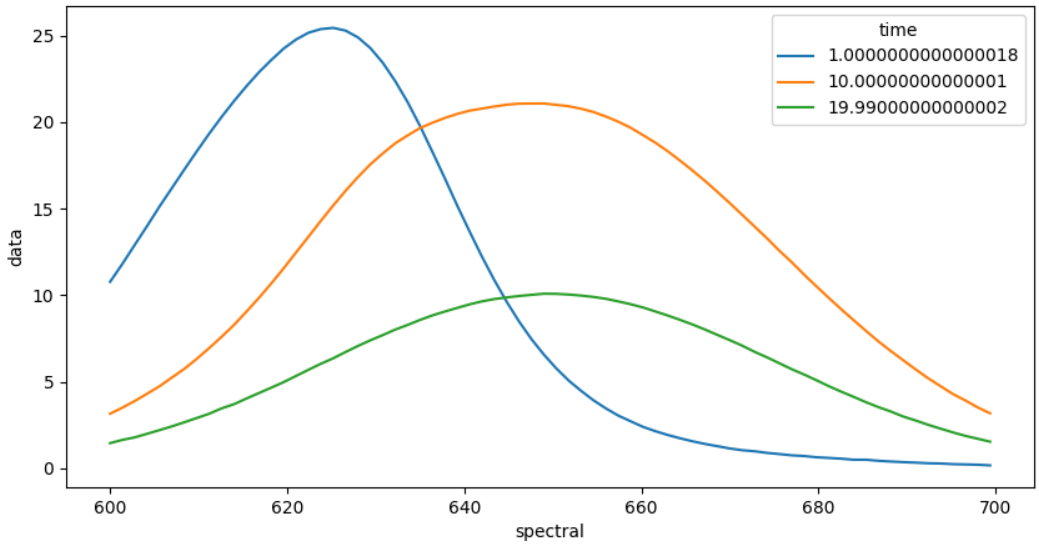
Now we can plot some time traces.

```
In [7]: plot_data = dataset.data.sel(spectral=[620, 630, 650], method='nearest')

In [8]: plot_data.plot.line(x='time', aspect=2, size=5);
```



We can also plot spectra at different times.

```
In [9]: plot_data = dataset.data.sel(time=[1, 10, 20], method='nearest')

In [10]: plot_data.plot.line(x='spectral', aspect=2, size=5);
```



To get an idea about how to model your data, you should inspect the singular value decomposition. Pyglotaran has a function to calculate it (among other things).

```
In [11]: dataset = gta.io.prepare_time_trace_dataset(dataset)

In [12]: dataset
Out[12]:
<xarray.Dataset>
```

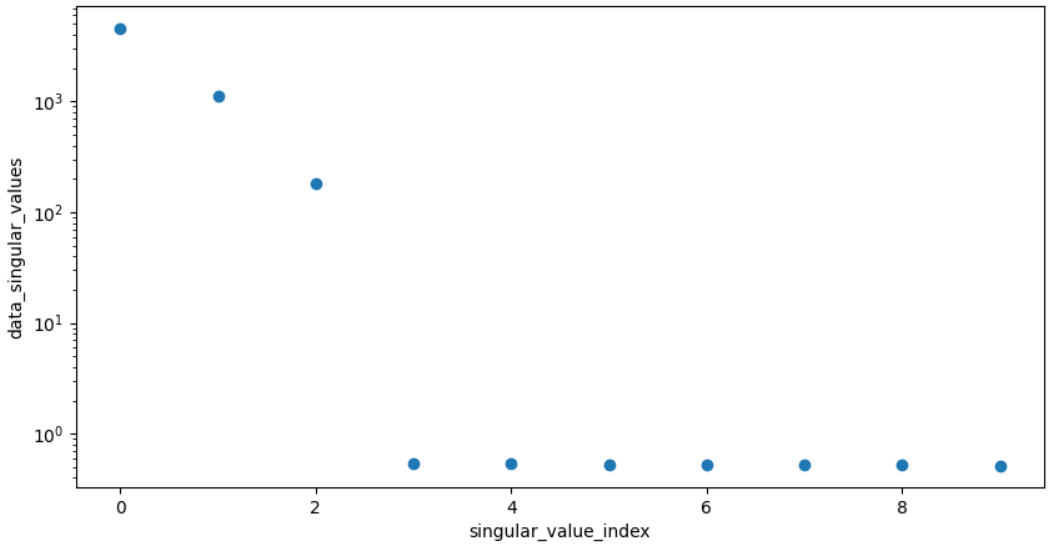
(continues on next page)

(continued from previous page)

```
Dimensions:                       (left_singular_value_index: 2100, right_singular_
↪value_index: 72, singular_value_index: 72, spectral: 72, time: 2100)
Coordinates:
  * time                          (time) float64 -1.0 -0.99 -0.98 ... 19.98 19.99
  * spectral                      (spectral) float64 600.0 601.4 ... 698.0 699.4
Dimensions without coordinates: left_singular_value_index, right_singular_value_index,
↪ singular_value_index
Data variables:
    data                          (time, spectral) float64 0.004229 ... 1.543
    data_left_singular_vectors    (time, left_singular_value_index) float64 -1...
    data_singular_values          (singular_value_index) float64 4.62e+03 ... ...
    data_right_singular_vectors   (right_singular_value_index, spectral) float64 ...
```

First, take a look at the first 10 singular values:

```
In [13]: plot_data = dataset.data_singular_values.sel(singular_value_index=range(0,␣
↪10))

In [14]: plot_data.plot(yscale='log', marker='o', linewidth=0, aspect=2, size=5);
```



To analyze our data, we need to create a model. Create a file called `model.py` in your working directory and fill it with the following:

```
type: kinetic-spectrum

initial_concentration:
  input:
    compartments: [s1, s2, s3]
    parameters: [input.1, input.0, input.0]

k_matrix:
  k1:
    matrix:
      (s2, s1): kinetic.1
```

(continues on next page)

(continued from previous page)

```
      (s3, s2): kinetic.2
      (s3, s3): kinetic.3

megacomplex:
  m1:
    k_matrix: [k1]

irf:
  irf1:
    type: gaussian
    center: irf.center
    width: irf.width

dataset:
  dataset1:
    initial_concentration: input
    megacomplex: [m1]
    irf: irf1
```

Now you can load the model file.

```
In [15]: model = gta.read_model_from_yaml_file('model.yml')
```

You can check your model for problems with `model.validate`.

```
In [16]: print(model.validate())
Your model is valid.
```

Now define some starting parameters. Create a file called `parameters.yml` with the following content.

```
input:
  - ['1', 1, {'vary': False, 'non-negative': False}]
  - ['0', 0, {'vary': False, 'non-negative': False}]

kinetic: [
     0.5,
     0.3,
     0.1,
]

irf:
  - ['center', 0.3]
  - ['width', 0.1]
```

```
In [17]: parameters = gta.read_parameters_from_yaml_file('parameters.yml')
```

You can `model.validate` also to check for missing parameters.

```
In [18]: print(model.validate(parameters=parameters))
Your model is valid.
```

Since not all problems in the model can be detected automatically it is wise to visually inspect the model. For this purpose, you can just print the model.

```
In [19]: print(model)
# Model
```

(continues on next page)

(continued from previous page)

```
_Type_: kinetic-spectrum

## Initial Concentration

* **input**:
  * *Label*: input
  * *Compartments*: ['s1', 's2', 's3']
  * *Parameters*: [input.1, input.0, input.0]
  * *Exclude From Normalize*: []

## K Matrix

* **k1**:
  * *Label*: k1
  * *Matrix*:
    * *('s2', 's1')*: kinetic.1
    * *('s3', 's2')*: kinetic.2
    * *('s3', 's3')*: kinetic.3


## Irf

* **irf1** (gaussian):
  * *Label*: irf1
  * *Type*: gaussian
  * *Center*: irf.center
  * *Width*: irf.width
  * *Normalize*: False
  * *Backsweep*: False

## Dataset

* **dataset1**:
  * *Label*: dataset1
  * *Megacomplex*: ['m1']
  * *Initial Concentration*: input
  * *Irf*: irf1

## Megacomplex

* **m1**:
  * *Label*: m1
  * *K Matrix*: ['k1']
```

The same way you should inspect your parameters.

```
In [20]: print(parameters)
* __None__:
  * __input__:
    * __1__: _Value_: 1.0, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: False, _
↪Non-Negative_: False, _Expr_: None
    * __0__: _Value_: 0.0, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: False, _
↪Non-Negative_: False, _Expr_: None
  * __kinetic__:
    * __1__: _Value_: 0.5, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: True, _Non-
↪Negative_: False, _Expr_: None
```

(continues on next page)

(continued from previous page)

```
    * __2__: _Value_: 0.3, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: True, _Non-
↪Negative_: False, _Expr_: None
    * __3__: _Value_: 0.1, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: True, _Non-
↪Negative_: False, _Expr_: None
  * __irf__:
    * __center__: _Value_: 0.3, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: True,␣
↪_Non-Negative_: False, _Expr_: None
    * __width__: _Value_: 0.1, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: True, _
↪Non-Negative_: False, _Expr_: None
```

Now we have everything together to optimize our parameters. First we import optimize.

```
In [21]: scheme = Scheme(model, parameters, {'dataset1': dataset})

In [22]: result = optimize(scheme)
   Iteration     Total nfev        Cost      Cost reduction    Step norm     ␣
↪Optimality
       0              1         7.5908e+00                                    1.
↪72e+01
       1              2         7.5907e+00      1.35e-04       1.07e-04       1.57e-
↪01
       2              3         7.5907e+00      2.24e-10       1.92e-08       6.62e-
↪06
`ftol` termination condition is satisfied.
Function evaluations 3, initial cost 7.5908e+00, final cost 7.5907e+00, first-order␣
↪optimality 6.62e-06.

In [23]: print(result)
Optimization Result             |            |
--------------------------------|------------|
 Number of residual evaluation |           3 |
           Number of variables |           5 |
          Number of datapoints |      151200 |
            Degrees of freedom |      151195 |
                    Chi Square |    1.52e+01 |
             Reduced Chi Square |    1.00e-04 |
 Root Mean Square Error (RMSE) |    1.00e-02 |


In [24]: print(result.optimized_parameters)
* __None__:
  * __input__:
    * __1__: _Value_: 1.0, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: False, _
↪Non-Negative_: False, _Expr_: None
    * __0__: _Value_: 0.0, _StdErr_: 0.0, _Min_: -inf, _Max_: inf, _Vary_: False, _
↪Non-Negative_: False, _Expr_: None
  * __kinetic__:
    * __1__: _Value_: 0.49990170497213715, _StdErr_: 0.0072604315075739745, _Min_: -
↪inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
    * __2__: _Value_: 0.3000421579328798, _StdErr_: 0.004195787756618939, _Min_: -inf,
↪ _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
    * __3__: _Value_: 0.09999809513191878, _StdErr_: 0.0004780267028974973, _Min_: -
↪inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
  * __irf__:
    * __center__: _Value_: 0.29999401098904216, _StdErr_: 0.0005010840353728522, _Min_
↪: -inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
    * __width__: _Value_: 0.0999983570632427, _StdErr_: 0.0006703827938132203, _Min_:␣
↪-inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
```

(continues on next page)

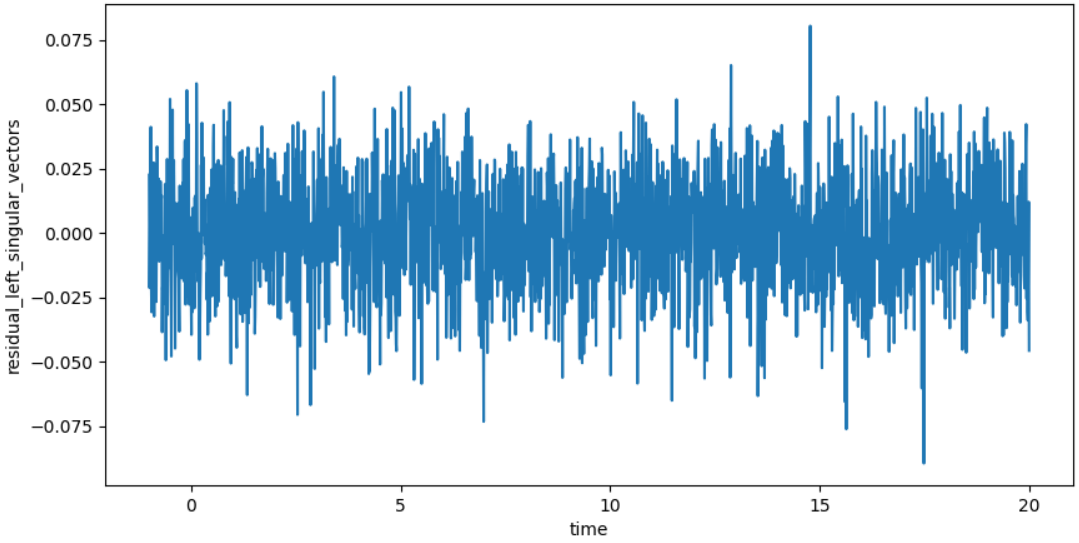
(continued from previous page)

You can get the resulting data for your dataset with `result.get_dataset`.

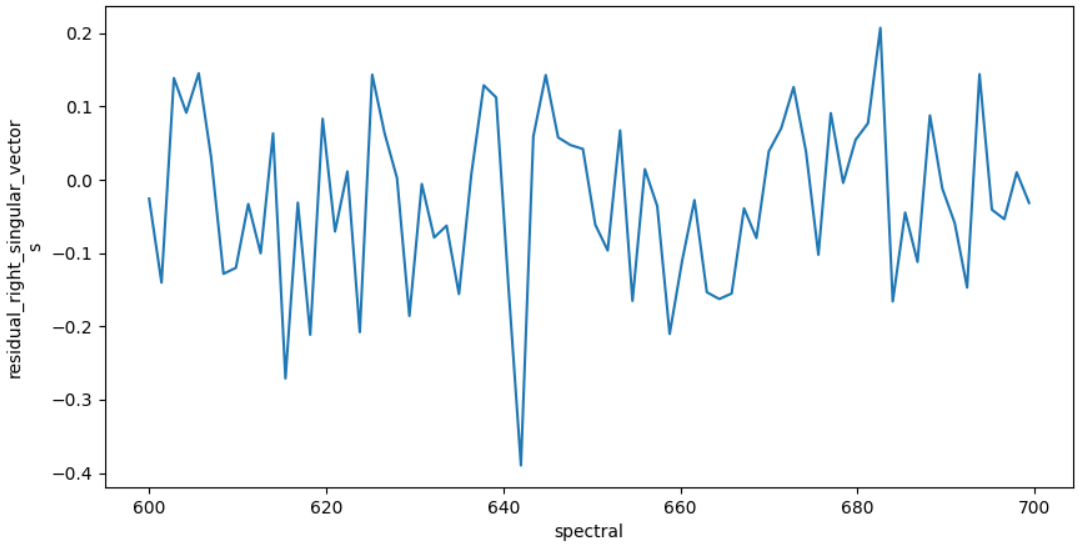
```
In [25]: result_dataset = result.get_dataset('dataset1')

In [26]: result_dataset
Out[26]:
<xarray.Dataset>
Dimensions:                                    (clp_label: 3, component: 3, from_
↪species: 3, left_singular_value_index: 2100, right_singular_value_index: 72,␣
↪singular_value_index: 72, species: 3, spectral: 72, time: 2100, to_species: 3)
Coordinates:
  * time                                       (time) float64 -1.0 ... 19.99
  * spectral                                   (spectral) float64 600.0 ... 699.4
  * clp_label                                  (clp_label) <U2 's1' 's2' 's3'
  * species                                    (species) <U2 's1' 's2' 's3'
    rate                                       (component) float64 -0.4999 ......
    lifetime                                   (component) float64 -2.0 ... -10.0
  * to_species                                 (to_species) <U2 's1' 's2' 's3'
  * from_species                               (from_species) <U2 's1' 's2' 's3'
Dimensions without coordinates: component, left_singular_value_index, right_singular_
↪value_index, singular_value_index
Data variables:
    data                                       (time, spectral) float64 0.0042...
    data_left_singular_vectors                 (time, left_singular_value_index)␣
↪float64 ...
    data_singular_values                       (singular_value_index) float64 ...
    data_right_singular_vectors                (spectral, right_singular_value_index)␣
↪float64 ...
    matrix                                     (time, clp_label) float64 6.082...
    clp                                        (spectral, clp_label) float64 1...
    weighted_residual                          (time, spectral) float64 0.0042...
    residual                                   (time, spectral) float64 0.0042...
    weighted_residual_left_singular_vectors    (time, left_singular_value_index)␣
↪float64 ...
    weighted_residual_right_singular_vectors   (right_singular_value_index, spectral)␣
↪float64 ...
    weighted_residual_singular_values          (singular_value_index) float64 ...
    residual_left_singular_vectors             (time, left_singular_value_index)␣
↪float64 ...
    residual_right_singular_vectors            (right_singular_value_index, spectral)␣
↪float64 ...
    residual_singular_values                   (singular_value_index) float64 ...
    fitted_data                                (time, spectral) float64 -4.416...
    species_associated_spectra                 (spectral, species) float64 15....
    species_concentration                      (time, species) float64 6.082e-...
    decay_associated_spectra                   (spectral, component) float64 2...
    a_matrix                                   (component, species) float64 1....
    k_matrix                                   (to_species, from_species) float64 ...
    k_matrix_reduced                           (to_species, from_species) float64 ...
    irf_center                                 float64 0.3
    irf_width                                  float64 0.1
    irf                                        (time) float64 2.001e-37 ... 0.0
Attributes:
    root_mean_square_error:           0.010020292428676811
    weighted_root_mean_square_error:  0.010020292428676811
```

The resulting data can be visualized the same way as the dataset. To judge the quality of the fit, you should look at first left and right singular vectors of the residual.

```
In [27]: plot_data = result_dataset.residual_left_singular_vectors.sel(left_singular_
↪value_index=0)

In [28]: plot_data.plot.line(x='time', aspect=2, size=5);
```




```
In [29]: plot_data = result_dataset.residual_right_singular_vectors.sel(right_
↪singular_value_index=0)

In [30]: plot_data.plot.line(x='spectral', aspect=2, size=5);
```



Finally, you can save your result.

```
In [31]: result_dataset.to_netcdf('dataset1.nc')
```

CHAPTER

FOUR

HISTORY

4.1 0.3.0 (2021-02-11)

- Significant code refactor with small API changes to parameter relation specification (see docs)
- Replaced lmfit with scipy.optimize

4.2 0.2.0 (2020-12-02)

- Large refactor with significant improvements but also small API changes (see docs)
- Removed doas plugin

4.3 0.1.0 (2020-07-14)

- Package was renamed to `pyglotaran` on PyPi

4.4 0.0.8 (2018-08-07)

- Changed `nan_policiy` to `omit`

4.5 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartement.

4.6 0.0.6 (2018-08-07)

- First release on PyPI, support for Windows installs added.
- Pre-Alpha Development

CHAPTER

FIVE

AUTHORS

5.1 Development Lead

- Joern Weissenborn <joern.weissenborn@gmail.com>
- Joris Snellenburg <j.snellenburg@gmail.com>

5.2 Contributors

- Sebastian Weigand <s.weigand.phy@gmail.com>

5.3 Special Thanks

- Stefan Schuetz
- Sergey P. Laptenok

5.4 Supervision

- **dr. Ivo H.M. van Stokkum** <i.h.m.van.stokkum@vu.nl> (VU University profile, personal homepage)

5.5 Original publications

1. Snellenburg JJ, Laptenok SP, Seger R, Mullen KM, van Stokkum IHM (2012). "Glotaran: A Java-Based Graphical User Interface for the R Package TIMP." Journal of Statistical Software, 49(3), 1–22. URL http://www.jstatsoft.org/v49/i03/.
2. Mullen, Katharine, & Ivo H. M. van Stokkum. "TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements." Journal of Statistical Software [Online], 18.3 (2007): 1 - 46. Web. 25 Jul. URL https://www.jstatsoft.org/article/view/v018i03
3. van Stokkum, IHM, Delmar S. Larsen, and Rienk van Grondelle. "Global and target analysis of time-resolved spectra." Biochimica et Biophysica Acta (BBA)-Bioenergetics 1657.2-3 (2004): 82-104. https://doi.org/10.1016/j.bbabio.2004.04.011

CHAPTER

SIX

OVERVIEW

CHAPTER

SEVEN

DATA IO

CHAPTER

EIGHT

PLOTTING

CHAPTER

NINE

MODELLING

CHAPTER

TEN

PARAMETER

CHAPTER

ELEVEN

OPTIMIZING

CHAPTER

TWELVE

API DOCUMENTATION

The API Documentation for pyglotaran is automatically created from its docstrings.

glotaran	Glotaran package __init__.py

12.1 glotaran

Glotaran package __init__.py

Modules

glotaran.analysis	This package contains functions for model simulation and fitting.
glotaran.builtin	
glotaran.cli	
glotaran.examples	
glotaran.io	Functions for data IO
glotaran.model	Glotaran Model Package
glotaran.parameter	
glotaran.parse	Glotarans parsing package

12.1.1 analysis

This package contains functions for model simulation and fitting.

Modules

glotaran.analysis.nnls	Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.
glotaran.analysis.optimize	
glotaran.analysis.problem	
glotaran.analysis.result	The result class for global analysis.
glotaran.analysis.scheme	
glotaran.analysis.simulation	Functions for simulating a global analysis model.
glotaran.analysis.variable_projection	Functions for calculating conditionally linear parameters and residual with the variable projection method.

nnls

Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.

Functions

Summary

residual_nnls	Calculate the conditionally linear parameters and residual with the nnls method.

residual_nnls

glotaran.analysis.nnls.**residual_nnls**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[List[str], numpy.ndarray]

Calculate the conditionally linear parameters and residual with the nnls method.

nnls stands for 'non-negative least-squares'.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

optimize

Functions

Summary

optimize	
optimize_problem	

optimize

glotaran.analysis.optimize.**optimize**(*scheme:* glotaran.analysis.scheme.Scheme, *verbose:* *bool* = *True*) → *glotaran.analysis.result.Result*

optimize_problem

glotaran.analysis.optimize.**optimize_problem**(*problem:* glotaran.analysis.problem.Problem, *verbose:* *bool* = *True*) → *glotaran.analysis.result.Result*

problem

Classes

Summary

GroupedProblem	
GroupedProblemDescriptor	
LabelAndMatrix	
Problem	A Problem class
ProblemDescriptor	

GroupedProblem

class glotaran.analysis.problem.**GroupedProblem**(*data*, *weight*, *has_scaling*, *group*, *data_sizes*, *descriptor*)

Bases: tuple

Create new instance of GroupedProblem(data, weight, has_scaling, group, data_sizes, descriptor)

Attributes Summary

data	Alias for field number 0
data_sizes	Holds the sizes of the concatenated datasets.
descriptor	Alias for field number 5
group	The concatenated labels of the involved datasets.
has_scaling	Indicates if at least one dataset in the group needs scaling.
weight	Alias for field number 1

data

GroupedProblem.**data**: **numpy.ndarray**

Alias for field number 0

data_sizes

GroupedProblem.**data_sizes**: **List[int]**

Holds the sizes of the concatenated datasets.

descriptor

GroupedProblem.**descriptor**: ***glotaran.analysis.problem.GroupedProblemDescriptor***

Alias for field number 5

group

GroupedProblem.**group**: **str**

The concatenated labels of the involved datasets.

has_scaling

GroupedProblem.**has_scaling**: **bool**
Indicates if at least one dataset in the group needs scaling.

weight

GroupedProblem.**weight**: **numpy.ndarray**
Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

GroupedProblem.**count**(*value*, */*)
Return number of occurrences of value.

index

GroupedProblem.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, */*)
Return number of occurrences of value.

data: **numpy.ndarray**
Alias for field number 0

data_sizes: **List[int]**
Holds the sizes of the concatenated datasets.

descriptor: ***glotaran.analysis.problem.GroupedProblemDescriptor***
Alias for field number 5

group: **str**
The concatenated labels of the involved datasets.

has_scaling: **bool**
Indicates if at least one dataset in the group needs scaling.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

weight: numpy.ndarray
Alias for field number 1

GroupedProblemDescriptor

class glotaran.analysis.problem.**GroupedProblemDescriptor**(*label*, *index*, *axis*)

Bases: tuple

Create new instance of GroupedProblemDescriptor(label, index, axis)

Attributes Summary

axis	Alias for field number 2
index	Alias for field number 1
label	Alias for field number 0

axis

GroupedProblemDescriptor.**axis:** numpy.ndarray
Alias for field number 2

index

GroupedProblemDescriptor.**index:** Any
Alias for field number 1

label

GroupedProblemDescriptor.**label:** str
Alias for field number 0

Methods Summary

count	Return number of occurrences of value.

count

GroupedProblemDescriptor.**count**(*value*, */*)
Return number of occurrences of value.

Methods Documentation

axis: numpy.ndarray
Alias for field number 2

count(*value*, */*)
Return number of occurrences of value.

index: Any
Alias for field number 1

label: str
Alias for field number 0

LabelAndMatrix

class glotaran.analysis.problem.**LabelAndMatrix**(*clp_label*, *matrix*)
Bases: tuple

Create new instance of LabelAndMatrix(clp_label, matrix)

Attributes Summary

clp_label	Alias for field number 0
matrix	Alias for field number 1

clp_label

LabelAndMatrix.**clp_label:** List[str]
Alias for field number 0

matrix

LabelAndMatrix.**matrix**: numpy.ndarray
Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

LabelAndMatrix.**count**(*value*, */*)
Return number of occurrences of value.

index

LabelAndMatrix.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

clp_label: List[str]
Alias for field number 0

count(*value*, */*)
Return number of occurrences of value.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

matrix: numpy.ndarray
Alias for field number 1

Problem

class glotaran.analysis.problem.**Problem**(*scheme:* glotaran.analysis.scheme.Scheme)

Bases: object

A Problem class

Initializes the Problem class from a scheme (*glotaran.analysis.scheme.Scheme*)

Args:

scheme (Scheme): An instance of *glotaran.analysis.scheme.Scheme* which defines your model, parameters, and data

Attributes Summary

additional_penalty	
bag	
clp_labels	
clps	
data	
filled_dataset_descriptors	
full_penalty	
grouped	
groups	
index_dependent	
matrices	
model	Property providing access to the used model
parameters	
reduced_clp_labels	
reduced_clps	
reduced_matrices	
residuals	
scheme	Property providing access to the used scheme
weighted_residuals	

additional_penalty

Problem.**additional_penalty**

bag

Problem.**bag**

clp_labels

Problem.**clp_labels**

clps

Problem.**clps**

data

Problem.**data**

filled_dataset_descriptors

Problem.**filled_dataset_descriptors**

full_penalty

Problem.**full_penalty**

grouped

Problem.**grouped**

groups

Problem.**groups**

index_dependent

Problem.**index_dependent**

matrices

Problem.**matrices**

model

Problem.**model**
Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*
Returns:
Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

parameters

Problem.**parameters**

reduced_clp_labels

Problem.**reduced_clp_labels**

reduced_clps

Problem.**reduced_clps**

reduced_matrices

Problem.**reduced_matrices**

residuals

Problem.**residuals**

scheme

Problem.**scheme**
Property providing access to the used scheme
Returns:
Scheme: An instance of *glotaran.analysis.scheme.Scheme* Provides access to data, model, parameters and optimization arguments.

weighted_residuals

Problem.**weighted_residuals**

Methods Summary

calculate_additional_penalty	Calculates additional penalties by calling the model.additional_penalty function.
calculate_index_dependent_grouped_matrices	
calculate_index_dependent_grouped_residual	
calculate_index_dependent_ungrouped_matrices	
calculate_index_dependent_ungrouped_residual	
calculate_index_independent_grouped_matrices	
calculate_index_independent_grouped_residual	
calculate_index_independent_ungrouped_matrices	
calculate_index_independent_ungrouped_residual	
calculate_matrices	
calculate_residual	
create_result_data	
reset	Resets all results and *DatasetDescriptors*.

calculate_additional_penalty

Problem.**calculate_additional_penalty**() → Union[numpy.ndarray, Dict[str, numpy.ndarray]]

Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_grouped_matrices

Problem.**calculate_index_dependent_grouped_matrices**() → Tuple[Dict[str, List[List[str]]], Dict[str, List[numpy.ndarray]], List[List[str]], List[numpy.ndarray]]

calculate_index_dependent_grouped_residual

Problem.**calculate_index_dependent_grouped_residual**() → Tuple[List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray]]

calculate_index_dependent_ungrouped_matrices

Problem.**calculate_index_dependent_ungrouped_matrices**() → Tuple[Dict[str, List[List[str]]], Dict[str, List[numpy.ndarray]], Dict[str, List[str]], Dict[str, List[numpy.ndarray]]]

calculate_index_dependent_ungrouped_residual

Problem.**calculate_index_dependent_ungrouped_residual**() → Tuple[Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]]]

calculate_index_independent_grouped_matrices

Problem.**calculate_index_independent_grouped_matrices**() → Tuple[Dict[str, List[str]], Dict[str, numpy.ndarray], Dict[str, *glotaran.analysis.problem.LabelAndMatrix*]]

calculate_index_independent_grouped_residual

Problem.**calculate_index_independent_grouped_residual**() → Tuple[List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray]]

calculate_index_independent_ungrouped_matrices

Problem.**calculate_index_independent_ungrouped_matrices**() → Tuple[Dict[str, List[str]], Dict[str, numpy.ndarray], Dict[str, List[str]], Dict[str, numpy.ndarray]]

calculate_index_independent_ungrouped_residual

Problem.**calculate_index_independent_ungrouped_residual**() → Tuple[Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]]]

calculate_matrices

Problem.**calculate_matrices**()

calculate_residual

Problem.**calculate_residual**()

create_result_data

Problem.**create_result_data**(*copy: bool = True*) → Dict[str, xarray.core.dataset.Dataset]

reset

Problem.**reset**()
Resets all results and *DatasetDescriptors*. Use after updating parameters.

Methods Documentation

property additional_penalty

property bag

calculate_additional_penalty() → Union[numpy.ndarray, Dict[str, numpy.ndarray]]
Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_grouped_matrices() → Tuple[Dict[str, List[List[str]]], Dict[str, List[numpy.ndarray]], List[List[str]], List[numpy.ndarray]]

calculate_index_dependent_grouped_residual() → Tuple[List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray]]

calculate_index_dependent_ungrouped_matrices() → Tuple[Dict[str, List[List[str]]], Dict[str, List[numpy.ndarray]], Dict[str, List[str]], Dict[str, List[numpy.ndarray]]]

calculate_index_dependent_ungrouped_residual() → Tuple[Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]]]

calculate_index_independent_grouped_matrices() → Tuple[Dict[str, List[str]], Dict[str, numpy.ndarray], Dict[str, *glotaran.analysis.problem.LabelAndMatrix*]]

calculate_index_independent_grouped_residual() → Tuple[List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray], List[numpy.ndarray]]

calculate_index_independent_ungrouped_matrices() → Tuple[Dict[str, List[str]], Dict[str, numpy.ndarray], Dict[str, List[str]], Dict[str, numpy.ndarray]]

calculate_index_independent_ungrouped_residual() → Tuple[Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]], Dict[str, List[numpy.ndarray]]]

calculate_matrices()

calculate_residual()

property clp_labels

property clps

create_result_data(*copy: bool = True*) → Dict[str, xarray.core.dataset.Dataset]

property data

property filled_dataset_descriptors

property full_penalty

property grouped

property groups

property index_dependent

property matrices

property model

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

property parameters

property reduced_clp_labels

property reduced_clps

property reduced_matrices

reset()

Resets all results and *DatasetDescriptors*. Use after updating parameters.

property residuals

property scheme

Property providing access to the used scheme

Returns:

Scheme: An instance of *glotaran.analysis.scheme.Scheme* Provides access to data, model, parameters and optimization arguments.

property weighted_residuals

ProblemDescriptor

class glotaran.analysis.problem.**ProblemDescriptor**(*dataset*, *data*, *model_axis*, *global_axis*, *weight*)

Bases: tuple

Create new instance of ProblemDescriptor(dataset, data, model_axis, global_axis, weight)

Attributes Summary

data	Alias for field number 1
dataset	Alias for field number 0
global_axis	Alias for field number 3
model_axis	Alias for field number 2
weight	Alias for field number 4

data

ProblemDescriptor.**data:** **xarray.core.dataarray.DataArray**

Alias for field number 1

dataset

ProblemDescriptor.**dataset:** *glotaran.model.dataset_descriptor.DatasetDescriptor*

Alias for field number 0

global_axis

ProblemDescriptor.**global_axis:** **numpy.ndarray**

Alias for field number 3

model_axis

ProblemDescriptor.**model_axis:** **numpy.ndarray**

Alias for field number 2

weight

ProblemDescriptor.**weight**: **xarray.core.dataarray.DataArray**
Alias for field number 4

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

ProblemDescriptor.**count**(*value*, */*)
Return number of occurrences of value.

index

ProblemDescriptor.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, */*)
Return number of occurrences of value.

data: **xarray.core.dataarray.DataArray**
Alias for field number 1

dataset: ***glotaran.model.dataset_descriptor.DatasetDescriptor***
Alias for field number 0

global_axis: **numpy.ndarray**
Alias for field number 3

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

model_axis: **numpy.ndarray**
Alias for field number 2

weight: **xarray.core.dataarray.DataArray**
Alias for field number 4

Exceptions

Exception Summary

ParameterError

ParameterError

exception glotaran.analysis.problem.**ParameterError**

result

The result class for global analysis.

Classes

Summary

Result	The result of a global analysis

Result

class glotaran.analysis.result.**Result**(*scheme: Scheme*, *data: dict[str, xr.Dataset]*, *optimized_parameters: ParameterGroup*, *additional_penalty: np.ndarray | None*, *least_squares_result: OptimizeResult*, *free_parameter_labels: list[str]*, *termination_reason: str*)

Bases: object

The result of a global analysis

Parameters

- **scheme** (Scheme) – An analysis scheme
- **data** (*Dict[str, xr.Dataset]*) – A dictionary containing all datasets with their labels as keys.
- **optimized_parameters** (ParameterGroup) – The optimized parameters, organized in a ParameterGroup
- **additional_penalty** (*Union[np.ndarray, None]*) – A vector with the value for each additional penalty, or None
- **least_squares_result** (*OptimizeResult*) – See scipy.optimize.OptimizeResult() scipy.optimize.least_squares()
- **free_parameter_labels** (*List[str]*) – The text labels of the free parameters that were optimized

- **termination_reason** (*str*) – The reason (message when) the optimizer terminated

Attributes Summary

additional_penalty	The additional penalty vector.
chi_square	The chi-square of the optimization.
covariance_matrix	Covariance matrix.
data	The resulting data as a dictionary of xarray.Dataset.
degrees_of_freedom	Degrees of freedom in optimization $N - N_{vars}$.
free_parameter_labels	List of labels of the free parameters used in optimization.
initial_parameters	The inititial parameters.
jacobian	Modified Jacobian matrix at the solution See also: scipy.optimize.least_squares()
model	The model for analysis.
nnls	If *True* non-negative least squares optimization is used instead of the default variable projection.
number_of_data_points	Number of data points N.
number_of_function_evaluations	The number of function evaluations.
number_of_jacobian_evaluations	The number of jacobian evaluations.
number_of_variables	Number of variables in optimization N_{vars}
optimized_parameters	The optimized parameters.
reduced_chi_square	The reduced chi-square of the optimization.
root_mean_square_error	The root mean square error the optimization.
scheme	The scheme for analysis.
success	Indicates if the optimization was successful.
termination_reason	The reason of the termination of the process.

additional_penalty

Result.**additional_penalty**
The additional penalty vector.

chi_square

Result.**chi_square**
The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

covariance_matrix

Result.**covariance_matrix**
Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

data

Result.**data**
The resulting data as a dictionary of xarray.Dataset.

Notes

The actual content of the data depends on the actual model and can be found in the documentation for the model.

degrees_of_freedom

Result.**degrees_of_freedom**
Degrees of freedom in optimization $N - N_{vars}$.

free_parameter_labels

Result.**free_parameter_labels**
List of labels of the free parameters used in optimization.

initial_parameters

Result.**initial_parameters**
The initital parameters.

jacobian

Result.**jacobian**
Modified Jacobian matrix at the solution See also: scipy.optimize.least_squares()
Returns Numpy array
Return type np.ndarray

model

Result.**model**
The model for analysis.

nnls

Result.**nnls**
If *True* non-negative least squares optimization is used instead of the default variable projection.

number_of_data_points

Result.**number_of_data_points**
Number of data points N.

number_of_function_evaluations

Result.**number_of_function_evaluations**
The number of function evaluations.

number_of_jacobian_evaluations

Result.**number_of_jacobian_evaluations**
The number of jacobian evaluations.

number_of_variables

Result.**number_of_variables**
Number of variables in optimization N_{vars}

optimized_parameters

Result.**optimized_parameters**
The optimized parameters.

reduced_chi_square

Result.**reduced_chi_square**
The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

root_mean_square_error

Result.**root_mean_square_error**
The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

scheme

Result.**scheme**
The scheme for analysis.

success

Result.**success**
Indicates if the optimization was successful.

termination_reason

Result.**termination_reason**
The reason of the termination of the process.

Methods Summary

get_dataset	Returns the result dataset for the given dataset label.
get_scheme	Return a new scheme from the Result object with optimized parameters.
markdown	Formats the model as a markdown text.
save	Saves the result to given folder.

get_dataset

Result.**get_dataset**(*dataset_label: str*) → xarray.core.dataset.Dataset
Returns the result dataset for the given dataset label.
Parameters **dataset_label** – The label of the dataset.

get_scheme

Result.**get_scheme**() → *glotaran.analysis.scheme.Scheme*
Return a new scheme from the Result object with optimized parameters.
Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.
Return type *Scheme*

markdown

Result.**markdown**(*with_model=True*) → str
Formats the model as a markdown text.

Parameters **with_model** – If *True*, the model will be printed with initial and optimized parameters filled in.

save

Result.**save**(*path: str*) → list[str]
Saves the result to given folder.

Returns a list with paths of all saved items.

The following files are saved:

- *result.md*: The result with the model formatted as markdown text.
- *optimized_parameters.csv*: The optimized parameter as csv file.
- *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters **path** – The path to the folder in which to save the result.

Methods Documentation

property additional_penalty
The additional penalty vector.

property chi_square
The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

property covariance_matrix
Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

property data
The resulting data as a dictionary of xarray.Dataset.

Notes

The actual content of the data depends on the actual model and can be found in the documentation for the model.

property degrees_of_freedom
Degrees of freedom in optimization $N - N_{vars}$.

property free_parameter_labels
List of labels of the free parameters used in optimization.

get_dataset(*dataset_label: str*) → xarray.core.dataset.Dataset
Returns the result dataset for the given dataset label.

Parameters **dataset_label** – The label of the dataset.

get_scheme() → *glotaran.analysis.scheme.Scheme*
Return a new scheme from the Result object with optimized parameters.

Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.
Return type *Scheme*

property initial_parameters
The initital parameters.

property jacobian
Modified Jacobian matrix at the solution See also: `scipy.optimize.least_squares()`
Returns Numpy array
Return type np.ndarray

markdown (*with_model=True*) → str
Formats the model as a markdown text.
Parameters **with_model** – If *True*, the model will be printed with initial and optimized parameters filled in.

property model
The model for analysis.

property nnls
If *True* non-negative least squares optimization is used instead of the default variable projection.

property number_of_data_points
Number of data points N.

property number_of_function_evaluations
The number of function evaluations.

property number_of_jacobian_evaluations
The number of jacobian evaluations.

property number_of_variables
Number of variables in optimization N_{vars}

property optimized_parameters
The optimized parameters.

property reduced_chi_square
The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

property root_mean_square_error
The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

save (*path: str*) → list[str]
Saves the result to given folder.

Returns a list with paths of all saved items.

The following files are saved:
- *result.md*: The result with the model formatted as markdown text.
- *optimized_parameters.csv*: The optimized parameter as csv file.
- *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters **path** – The path to the folder in which to save the result.

property scheme
The scheme for analysis.

property success
Indicates if the optimization was successful.

property termination_reason
The reason of the termination of the process.

scheme

Classes

Summary

Scheme

Scheme

class glotaran.analysis.scheme.**Scheme**(*model: Model = None, parameters: ParameterGroup = None, data: dict[str, xr.DataArray | xr.Dataset] = None, group_tolerance: float = 0.0, non_negative_least_squares: bool = False, maximum_number_function_evaluations: int = None, ftol: float = 1e-08, gtol: float = 1e-08, xtol: float = 1e-08, optimization_method: Literal[TrustRegionReflection, Dogbox, Levenberg-Marquardt] = 'TrustRegionReflection'*)

Bases: object

Attributes Summary

data
ftol
group_tolerance
gtol
maximum_number_function_evaluations
model
non_negative_least_squares

continues on next page

Table 22 – continued from previous page

optimization_method
parameters
xtol

data

Scheme.**data**

ftol

Scheme.**ftol**

group_tolerance

Scheme.**group_tolerance**

gtol

Scheme.**gtol**

maximum_number_function_evaluations

Scheme.**maximum_number_function_evaluations**

model

Scheme.**model**

non_negative_least_squares

Scheme.**non_negative_least_squares**

optimization_method

Scheme.**optimization_method**

parameters

Scheme.**parameters**

xtol

Scheme.**xtol**

Methods Summary

from_yaml_file	
markdown	
problem_list	Returns a list with all problems in the model and missing parameters.
valid	Returns *True* if there are no problems with the model or the parameters, else *False*.
validate	Returns a string listing all problems in the model and missing parameters.

from_yaml_file

classmethod Scheme.**from_yaml_file**(*filename: str*) → *glotaran.analysis.scheme.Scheme*

markdown

Scheme.**markdown**()

problem_list

Scheme.**problem_list**() → list[str]

Returns a list with all problems in the model and missing parameters.

valid

Scheme.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if there are no problems with the model or the parameters, else *False*.

validate

Scheme.**validate**() → str
Returns a string listing all problems in the model and missing parameters.

Methods Documentation

property data

classmethod from_yaml_file(*filename: str*) → *glotaran.analysis.scheme.Scheme*

property ftol

property group_tolerance

property gtol

markdown()

property maximum_number_function_evaluations

property model

property non_negative_least_squares

property optimization_method

property parameters

problem_list() → list[str]
Returns a list with all problems in the model and missing parameters.

valid(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*]* = *None*) → bool
Returns *True* if there are no problems with the model or the parameters, else *False*.

validate() → str
Returns a string listing all problems in the model and missing parameters.

property xtol

simulation

Functions for simulating a global analysis model.

Functions

Summary

simulate	Simulates a model.

simulate

glotaran.analysis.simulation.**simulate**(*model: Model*, *dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise=False*, *noise_std_dev=1.0*, *noise_seed=None*)

Simulates a model.

Parameters

- **model** – The model to simulate.
- **parameter** – The parameters for the simulation.
- **dataset** – Label of the dataset to simulate
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Will be used instead of *model.global_matrix* if given.
- **noise** – Add noise to the simulation.
- **noise_std_dev** – The standard deviation for noise simulation.
- **noise_seed** – The seed for the noise simulation.

variable_projection

Functions for calculating conditionally linear parameters and residual with the variable projection method.

Functions

Summary

residual_variable_projection	Calculates the conditionally linear parameters and residual with the variable projection method.

residual_variable_projection

glotaran.analysis.variable_projection.**residual_variable_projection**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[List[str], numpy.ndarray]

Calculates the conditionally linear parameters and residual with the variable projection method.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

12.1.2 builtin

Modules

glotaran.builtin.file_formats	
glotaran.builtin.models	Glotaran Models Package

file_formats

Modules

glotaran.builtin.file_formats.ascii	
glotaran.builtin.file_formats.sdt	

ascii

Modules

glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file	

wavelength_time_explicit_file

Functions

Summary

get_data_file_format	
get_interval_number	
read_ascii_time_trace	Reads an ascii file in wavelength- or time-explicit format.
write_ascii_time_trace	

get_data_file_format

glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**get_data_file_format**

get_interval_number

glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**get_interval_number**(

read_ascii_time_trace

glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**read_ascii_time_trac**

Reads an ascii file in wavelength- or time-explicit format.

See [1] for documentation of this format.

Parameters **fname** (*str*) – Name of the ascii file.

Returns **dataset**

Return type xr.Dataset

Notes

write_ascii_time_trace

glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**write_ascii_time_tra**

Classes

Summary

DataFileType	An enumeration.
ExplicitFile	Abstract class representing either a time- or wavelength-explicit file.
TimeExplicitFile	Represents a time explicit file
WavelengthExplicitFile	Represents a wavelength explicit file

DataFileType

class glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**DataFileType**(*va*

Bases: enum.Enum

An enumeration.

Attributes Summary

time_explicit
wavelength_explicit

time_explicit

DataFileType.**time_explicit = 'Time explicit'**

wavelength_explicit

DataFileType.**wavelength_explicit = 'Wavelength explicit'**

time_explicit = 'Time explicit'

wavelength_explicit = 'Wavelength explicit'

ExplicitFile

class glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**ExplicitFile**(*fil*
O
ti
=
N
d
O
ti
=
N

Bases: object

Abstract class representing either a time- or wavelength-explicit file.

Methods Summary

dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

dataset

ExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

ExplicitFile.**get_data_row**(*index*)

get_explicit_axis

ExplicitFile.**get_explicit_axis**()

get_format_name

ExplicitFile.**get_format_name**()

get_observations

ExplicitFile.**get_observations**(*index*)

get_secondary_axis

ExplicitFile.**get_secondary_axis**()

read

ExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

ExplicitFile.**set_explicit_axis**(*axis*)

write

ExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

Methods Documentation

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

TimeExplicitFile

class glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**TimeExplicitFi**

Bases: *glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a time explicit file

Methods Summary

add_data_row
dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

add_data_row

TimeExplicitFile.**add_data_row**(*row*)

dataset

TimeExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

TimeExplicitFile.**get_data_row**(*index*)

get_explicit_axis

TimeExplicitFile.**get_explicit_axis**()

get_format_name

TimeExplicitFile.**get_format_name**()

get_observations

TimeExplicitFile.**get_observations**(*index*)

get_secondary_axis

TimeExplicitFile.**get_secondary_axis**()

read

TimeExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

TimeExplicitFile.**set_explicit_axis**(*axies*)

write

TimeExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axies*)

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

WavelengthExplicitFile

class glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.**WavelengthExpli**

Bases: *glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a wavelength explicit file

Methods Summary

add_data_row
dataset
get_data_row

continues on next page

Table 34 – continued from previous page

get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
times
wavelengths
write

add_data_row

WavelengthExplicitFile.**add_data_row**(*row*)

dataset

WavelengthExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

WavelengthExplicitFile.**get_data_row**(*index*)

get_explicit_axis

WavelengthExplicitFile.**get_explicit_axis**()

get_format_name

WavelengthExplicitFile.**get_format_name**()

get_observations

WavelengthExplicitFile.**get_observations**(*index*)

get_secondary_axis

WavelengthExplicitFile.**get_secondary_axis**()

read

WavelengthExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

WavelengthExplicitFile.**set_explicit_axis**(*axis*)

times

WavelengthExplicitFile.**times**()

wavelengths

WavelengthExplicitFile.**wavelengths**()

write

WavelengthExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

times()

wavelengths()

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

sdt

Modules

glotaran.builtin.file_formats.sdt.sdt_file_reader	Glotarans module to read files

sdt_file_reader

Glotarans module to read files

Functions

Summary

read_sdt	Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

read_sdt

glotaran.builtin.file_formats.sdt.sdt_file_reader.**read_sdt**(*file_path: str*, *index: Optional[numpy.ndarray] = None*, *flim: bool = False*, *dataset_index: Optional[int] = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → xarray.core.dataset.Dataset

Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

Parameters

- **file_path** (*str*) – Path to the sdt file which should be read.
- **index** (*list, np.ndarray*) – This is only needed if *type_of_data=="st"*, since **.sdt* files, which only contain spectral temporal data, lack the spectral information. Thus for the spectral axis data need to be given by the user.
- **flim** – Set true if reading a result from a FLIM measurement.
- **dataset_index** (*int: default 0*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one
- **swap_axis** (*bool, default False*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset.
- **orig_time_axis_index** (*int*) – Index of the axis which corresponds to the time axis. I.e. for data of shape (64, 64, 256), which are a 64x64 pixel map with 256 time steps, orig_time_axis_index=2.

Raises **IndexError:** – If the length of the index array is incompatible with the data.

models

Glotaran Models Package

Modules

glotaran.builtin.models.kinetic_image
glotaran.builtin.models.kinetic_spectrum

kinetic_image

Modules

glotaran.builtin.models.kinetic_image.initial_concentration	This package contains the initial concentration item.
glotaran.builtin.models.kinetic_image.irf	This package contains irf items.
glotaran.builtin.models.kinetic_image.k_matrix	K-Matrix
glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor	Kinetic Image Dataset Descriptor
glotaran.builtin.models.kinetic_image.kinetic_image_matrix	Glotaran Kinetic Matrix
glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex	This package contains the kinetic megacomplex item.
glotaran.builtin.models.kinetic_image.kinetic_image_model	

continues on next page

Table 38 – continued from previous page

glotaran.builtin.models.kinetic_image.kinetic_image_result

initial_concentration

This package contains the initial concentration item.

Classes

Summary

InitialConcentration	An initial concentration describes the population of the compartments at the beginning of an experiment.

InitialConcentration

class glotaran.builtin.models.kinetic_image.initial_concentration.**InitialConcentration**

Bases: object

An initial concentration describes the population of the compartments at the beginning of an experiment.

Attributes Summary

compartments
exclude_from_normalize
label
parameters

compartments

InitialConcentration.**compartments**

exclude_from_normalize

InitialConcentration.**exclude_from_normalize**

label

InitialConcentration.**label**

parameters

InitialConcentration.**parameters**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
normalized	
validate	

fill

InitialConcentration.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod InitialConcentration.**from_dict**(*values: dict*) → cls

from_list

classmethod InitialConcentration.**from_list**(*values: list*) → cls

mprint

InitialConcentration.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

normalized

InitialConcentration.**normalized**(*dataset:* glotaran.model.dataset_descriptor.DatasetDescriptor) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialConce*

validate

InitialConcentration.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property compartments

property exclude_from_normalize

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

normalized(*dataset:* glotaran.model.dataset_descriptor.DatasetDescriptor) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*

property parameters

validate(*model: Model*, *parameters=None*) → list[str]

irf

This package contains irf items.

Classes

Summary

Irf	Represents an IRF.
IrfGaussian	
IrfMeasured	A measured IRF.
IrfMultiGaussian	Represents a gaussian IRF.

Irf

class glotaran.builtin.models.kinetic_image.irf.**Irf**

Bases: object

Represents an IRF.

Methods Summary

add_type

add_type

classmethod Irf.**add_type**(*type_name: str*, *attribute_type: type*)

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

IrfGaussian

class glotaran.builtin.models.kinetic_image.irf.**IrfGaussian**

Bases: *glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian*

Attributes Summary

backsweep
backsweep_period
center
label
normalize
scale
type
width

backsweep

IrfGaussian.**backsweep**

backsweep_period

IrfGaussian.**backsweep_period**

center

IrfGaussian.**center**

label

IrfGaussian.**label**

normalize

IrfGaussian.**normalize**

scale

IrfGaussian.**scale**

type

IrfGaussian.**type**

width

IrfGaussian.**width**

Methods Summary

calculate	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfGaussian.**calculate**(*index*, *axis*)

fill

IrfGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfGaussian.**from_list**(*values: list*) → cls

mprint

IrfGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfGaussian.**parameter**(*index*)

validate

IrfGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep

property backsweep_period

calculate(*index*, *axis*)

property center

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize

parameter(*index*)

property scale

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

IrfMeasured

class glotaran.builtin.models.kinetic_image.irf.**IrfMeasured**
Bases: object

A measured IRF. The data must be supplied by the dataset.

Attributes Summary

label	
type	

label

IrfMeasured.**label**

type

IrfMeasured.**type**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

fill

IrfMeasured.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfMeasured.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfMeasured.**from_list**(*values: list*) → cls

mprint

IrfMeasured.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

IrfMeasured.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type

validate(*model: Model*, *parameters=None*) → list[str]

IrfMultiGaussian

class glotaran.builtin.models.kinetic_image.irf.**IrfMultiGaussian**
Bases: object

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Attributes Summary

backsweep
backsweep_period
center
label
normalize
scale
type
width

backsweep

IrfMultiGaussian.**backsweep**

backsweep_period

IrfMultiGaussian.**backsweep_period**

center

IrfMultiGaussian.**center**

label

IrfMultiGaussian.**label**

normalize

IrfMultiGaussian.**normalize**

scale

IrfMultiGaussian.**scale**

type

IrfMultiGaussian.**type**

width

IrfMultiGaussian.**width**

Methods Summary

calculate	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	

continues on next page

Table 49 – continued from previous page

mprint
parameter
validate

calculate

`IrfMultiGaussian.`**`calculate`**(*index*, *axis*)

fill

`IrfMultiGaussian.`**`fill`**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **`model`** – A glotaran model.
- **`parameter`** (`ParameterGroup`) – The parameter group to fill from.

from_dict

`classmethod` `IrfMultiGaussian.`**`from_dict`**(*values: dict*) → cls

from_list

`classmethod` `IrfMultiGaussian.`**`from_list`**(*values: list*) → cls

mprint

`IrfMultiGaussian.`**`mprint`**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

`IrfMultiGaussian.`**`parameter`**(*index*)

validate

IrfMultiGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep

property backsweep_period

calculate(*index*, *axis*)

property center

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize

parameter(*index*)

property scale

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

k_matrix

K-Matrix

Classes

Summary

KMatrix	A K-Matrix represents a first order differental system.

KMatrix

class glotaran.builtin.models.kinetic_image.k_matrix.**KMatrix**
Bases: object

A K-Matrix represents a first order differental system.

Attributes Summary

label	
matrix	

label

KMatrix.**label**

matrix

KMatrix.**matrix**

Methods Summary

a_matrix	The resulting A matrix of the KMatrix.
a_matrix_as_markdown	Returns the A Matrix as markdown formatted table.
a_matrix_non_unibranch	The resulting A matrix of the KMatrix for a non-unibranched model.
a_matrix_unibranch	The resulting A matrix of the KMatrix for an unibranched model.
combine	Creates a combined matrix.
eigen	Returns the eigenvalues and eigenvectors of the k matrix.
empty	Creates an empty K-Matrix.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
full	The full representation of the KMatrix as numpy array.
involved_compartments	A list of all compartments in the Matrix.
is_unibranched	Returns true in the KMatrix represents an unibranched model.

continues on next page

Table 52 – continued from previous page

matrix_as_markdown	Returns the KMatrix as markdown formatted table.
mprint	
rates	The resulting rates of the matrix.
reduced	The reduced representation of the KMatrix as numpy array.
validate	

a_matrix

KMatrix.**a_matrix**(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentra
→ numpy.ndarray

The resulting A matrix of the KMatrix.

Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown

KMatrix.**a_matrix_as_markdown**(*initial_concentration:*
glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration
→ str

Returns the A Matrix as markdown formatted table.

Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch

KMatrix.**a_matrix_non_unibranch**(*initial_concentration:*
glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentrati
→ numpy.ndarray

The resulting A matrix of the KMatrix for a non-unibranched model.

Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch

KMatrix.**a_matrix_unibranch**(*initial_concentration:*
glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration)
→ numpy.array

The resulting A matrix of the KMatrix for an unibranched model.

Parameters **initial_concentration** – The initial concentration.

combine

KMatrix.**combine**(*k_matrix:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

Creates a combined matrix.

Parameters **k_matrix** – KMatrix to combine with.
Returns The combined KMatrix.
Return type combined

eigen

KMatrix.**eigen**(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]

Returns the eigenvalues and eigenvectors of the k matrix.

Parameters **compartments** – The compartment order.

empty

classmethod KMatrix.**empty**(*label: str*, *compartments: list[str]*) → KMatrix

Creates an empty K-Matrix. Useful for combining.

Parameters

- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill

KMatrix.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KMatrix.**from_dict**(*values: dict*) → cls

from_list

classmethod KMatrix.**from_list**(*values: list*) → cls

full

KMatrix.**full**(*compartments: list[str]*) → np.ndarray
The full representation of the KMatrix as numpy array.
Parameters **compartments** – The compartment order.

involved_compartments

KMatrix.**involved_compartments**() → list[str]
A list of all compartments in the Matrix.

is_unibranched

KMatrix.**is_unibranched**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialC*
→ bool
Returns true in the KMatrix represents an unibranched model.
Parameters **initial_concentration** – The initial concentration.

matrix_as_markdown

KMatrix.**matrix_as_markdown**(*compartments: list[str] = None*, *fill_parameters: bool = False*) → str
Returns the KMatrix as markdown formatted table.
Parameters
- **compartments** – (default = None) An optional list defining the desired order of compartments.
- **fill_parameters** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

mprint

KMatrix.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

rates

KMatrix.**rates**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*
→ numpy.ndarray
The resulting rates of the matrix.
Parameters **initial_concentration** – The initial concentration.

reduced

KMatrix.**reduced**(*compartments: list[str]*) → np.ndarray
The reduced representation of the KMatrix as numpy array.
Parameters **compartments** – The compartment order.

validate

KMatrix.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

a_matrix(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration) → numpy.ndarray
The resulting A matrix of the KMatrix.
Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialCon → str
Returns the A Matrix as markdown formatted table.
Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialC → numpy.ndarray
The resulting A matrix of the KMatrix for a non-unibranched model.
Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcen → numpy.array
The resulting A matrix of the KMatrix for an unibranched model.
Parameters **initial_concentration** – The initial concentration.

combine(*k_matrix:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*
Creates a combined matrix.
Parameters **k_matrix** – KMatrix to combine with.
Returns The combined KMatrix.
Return type combined

eigen(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]
Returns the eigenvalues and eigenvectors of the k matrix.
Parameters **compartments** – The compartment order.

classmethod **empty**(*label: str*, *compartments: list[str]*) → *KMatrix*
Creates an empty K-Matrix. Useful for combining.
Parameters
- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

full(*compartments: list[str]*) → np.ndarray
The full representation of the KMatrix as numpy array.
Parameters **compartments** – The compartment order.

involved_compartments() → list[str]
A list of all compartments in the Matrix.

is_unibranched(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentratio
→ bool
Returns true in the KMatrix represents an unibranched model.
Parameters **initial_concentration** – The initial concentration.

property label

property matrix

matrix_as_markdown(*compartments: list[str] = None*, *fill_parameters: bool = False*)
→ str
Returns the KMatrix as markdown formatted table.
Parameters
- **compartments** – (default = None) An optional list defining the desired order of compartments.
- **fill_parameters** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

rates(*initial_concentration:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration)
→ numpy.ndarray
The resulting rates of the matrix.
Parameters **initial_concentration** – The initial concentration.

reduced(*compartments: list[str]*) → np.ndarray
The reduced representation of the KMatrix as numpy array.
Parameters **compartments** – The compartment order.

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_image_dataset_descriptor

Kinetic Image Dataset Descriptor

Classes

Summary

KineticImageDatasetDescriptor

KineticImageDatasetDescriptor

class glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.**KineticIma**
Bases: *glotaran.model.dataset_descriptor.DatasetDescriptor*

Attributes Summary

baseline
initial_concentration
irf
label
megacomplex
scale

baseline

KineticImageDatasetDescriptor.**baseline**

initial_concentration

KineticImageDatasetDescriptor.**initial_concentration**

irf

KineticImageDatasetDescriptor.**irf**

label

KineticImageDatasetDescriptor.**label**

megacomplex

KineticImageDatasetDescriptor.**megacomplex**

scale

KineticImageDatasetDescriptor.**scale**

Methods Summary

compartments	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
get_k_matrices	
mprint	
validate	

compartments

KineticImageDatasetDescriptor.**compartments**()

fill

KineticImageDatasetDescriptor.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KineticImageDatasetDescriptor.**from_dict**(*values: dict*) → cls

from_list

classmethod KineticImageDatasetDescriptor.**from_list**(*values: list*) → cls

get_k_matrices

KineticImageDatasetDescriptor.**get_k_matrices**()

mprint

KineticImageDatasetDescriptor.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

KineticImageDatasetDescriptor.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property baseline

compartments()

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

get_k_matrices()

property initial_concentration

property irf

property label

property megacomplex

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_image_matrix

Glotaran Kinetic Matrix

Functions

Summary

calculate_kinetic_matrix_gaussian_irf	Calculates a kinetic matrix with a gaussian irf.
calculate_kinetic_matrix_no_irf	
kinetic_image_matrix_implementation	
kinetic_matrix	

calculate_kinetic_matrix_gaussian_irf

glotaran.builtin.models.kinetic_image.kinetic_image_matrix.**calculate_kinetic_matrix_ga**

Calculates a kinetic matrix with a gaussian irf.

calculate_kinetic_matrix_no_irf

glotaran.builtin.models.kinetic_image.kinetic_image_matrix.**calculate_kinetic_matrix_no**

kinetic_image_matrix_implementation

glotaran.builtin.models.kinetic_image.kinetic_image_matrix.**kinetic_image_matrix_implem**

kinetic_matrix

glotaran.builtin.models.kinetic_image.kinetic_image_matrix.**kinetic_matrix**(*dataset_descriptor*, *axis=None*, *index=None*, *irf=None*, *matrix_implementation*

kinetic_image_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

KineticImageMegacomplex	A Megacomplex with one or more K-Matrices.

KineticImageMegacomplex

class glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.**KineticImageMega**

Bases: object

A Megacomplex with one or more K-Matrices.

Attributes Summary

involved_compartments	
k_matrix	
label	
scale	

involved_compartments

KineticImageMegacomplex.**involved_compartments**

k_matrix

KineticImageMegacomplex.**k_matrix**

label

KineticImageMegacomplex.**label**

scale

KineticImageMegacomplex.**scale**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
full_k_matrix	
mprint	
validate	

fill

KineticImageMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KineticImageMegacomplex.**from_dict**(*values: dict*) → cls

from_list

classmethod KineticImageMegacomplex.**from_list**(*values: list*) → cls

full_k_matrix

KineticImageMegacomplex.**full_k_matrix**(*model=None*)

mprint

KineticImageMegacomplex.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

KineticImageMegacomplex.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

full_k_matrix(*model=None*)

property involved_compartments

property k_matrix

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_image_model

Functions

Summary

kinetic_image_matrix

kinetic_image_matrix

glotaran.builtin.models.kinetic_image.kinetic_image_model.**kinetic_image_matrix**(*dataset_de*
axis=None
in-
dex=None,
irf=None)

Classes

Summary

KineticImageModel

KineticImageModel

class glotaran.builtin.models.kinetic_image.kinetic_image_model.**KineticImageModel**
Bases: *glotaran.model.base_model.Model*

Attributes Summary

additional_penalty_function
constrain_matrix_function
dataset
global_dimension
global_matrix
has_additional_penalty_function
has_matrix_constraints_function

continues on next page

Table 62 – continued from previous page

index_dependent_matrix	
initial_concentration	
irf	
k_matrix	
megacomplex	
model_dimension	
model_type	The type of the model as human readable string.
retrieve_clp_function	
weights	

additional_penalty_function

KineticImageModel.**additional_penalty_function = None**

constrain_matrix_function

KineticImageModel.**constrain_matrix_function = None**

dataset

KineticImageModel.**dataset**

global_dimension

KineticImageModel.**global_dimension = 'pixel'**

global_matrix

KineticImageModel.**global_matrix = None**

has_additional_penalty_function

KineticImageModel.**has_additional_penalty_function = None**

has_matrix_constraints_function

KineticImageModel.**has_matrix_constraints_function = None**

index_dependent_matrix

KineticImageModel.**index_dependent_matrix**

initial_concentration

KineticImageModel.**initial_concentration**

irf

KineticImageModel.**irf**

k_matrix

KineticImageModel.**k_matrix**

megacomplex

KineticImageModel.**megacomplex**

model_dimension

KineticImageModel.**model_dimension = 'time'**

model_type

KineticImageModel.**model_type**
The type of the model as human readable string.

retrieve_clp_function

KineticImageModel.**retrieve_clp_function = None**

weights

KineticImageModel.**weights**

Methods Summary

add_weights	
finalize_data	
from_dict	Creates a model from a dictionary.
get_dataset	
get_initial_concentration	
get_irf	
get_k_matrix	
get_megacomplex	
grouped	
index_dependent	
markdown	Formats the model as Markdown string.
matrix	
problem_list	Returns a list with all problems in the model and missing parameters if specified.
set_dataset	
set_initial_concentration	
set_irf	
set_k_matrix	
set_megacomplex	
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

add_weights

KineticImageModel.**add_weights**(*item:* glotaran.model.weight.Weight)

finalize_data

KineticImageModel.**finalize_data**(*problem: Problem*, *data: dict[str, xr.Dataset]*)

from_dict

classmethod KineticImageModel.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset

KineticImageModel.**get_dataset**(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.Kineti*

get_initial_concentration

KineticImageModel.**get_initial_concentration**(*label: str*) → *glotaran.builtin.models.kinetic_image.initial_concentrati*

get_irf

KineticImageModel.**get_irf**(*label: str*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix

KineticImageModel.**get_k_matrix**(*label: str*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex

KineticImageModel.**get_megacomplex**(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.Kine*

grouped

KineticImageModel.**grouped**()

index_dependent

KineticImageModel.**index_dependent**()

markdown

KineticImageModel.**markdown**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*, *initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** – Parameter to include.
- **initial** – Initial values for the parameters.

matrix

static KineticImageModel.**matrix**(*dataset_descriptor=None*, *axis=None*, *index=None*, *irf=None*)

problem_list

KineticImageModel.**problem_list**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

set_dataset

KineticImageModel.**set_dataset**(*label: str*, *item: glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.Kineti*

set_initial_concentration

KineticImageModel.**set_initial_concentration**(*label: str*, *item: glotaran.builtin.models.kinetic_image.initial_concentratio*

set_irf

KineticImageModel.**set_irf**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix

KineticImageModel.**set_k_matrix**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex

KineticImageModel.**set_megacomplex**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.Kine

simulate

KineticImageModel.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

KineticImageModel.**valid**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate

KineticImageModel.**validate**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

Methods Documentation

add_weights(*item:* glotaran.model.weight.Weight)

additional_penalty_function = None

constrain_matrix_function = None

property dataset

finalize_data(*problem: Problem*, *data: dict[str, xr.Dataset]*)

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageData*

get_initial_concentration(*label: str*) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialCon*

get_irf(*label: str*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix(*label: str*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.KineticImageMeg*

global_dimension = 'pixel'

global_matrix = None

grouped()

has_additional_penalty_function = None

has_matrix_constraints_function = None

index_dependent()

property index_dependent_matrix

property initial_concentration

property irf

property k_matrix

markdown(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** – Parameter to include.
- **initial** – Initial values for the parameters.

static matrix(*dataset_descriptor=None*, *axis=None*, *index=None*, *irf=None*)

property megacomplex

model_dimension = 'time'

property model_type

The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

retrieve_clp_function = None

set_dataset(*label: str*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDat

set_initial_concentration(*label: str*, *item:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialCo

set_irf(*label: str*, *item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix(*label: str*, *item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex(*label: str*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.KineticImageM

simulate(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property weights

kinetic_image_result

Functions

Summary

finalize_kinetic_image_result
retrieve_decay_assocatiated_data
retrieve_irf
retrieve_species_assocatiated_data

finalize_kinetic_image_result

glotaran.builtin.models.kinetic_image.kinetic_image_result.**finalize_kinetic_image_resu**

retrieve_decay_assocatiated_data

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_decay_assocatiated_**

retrieve_irf

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_irf**(*model*, *dataset*, *dataset_descriptor*, *name*)

retrieve_species_assocatiated_data

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_species_assocatiat**

kinetic_spectrum

Modules

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor	
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_matrix	Glotaran kinetic spectrum Matrix
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model	
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result	

continues on next page

Table 65 – continued from previous page

glotaran.builtin.models.kinetic_spectrum.spectral_constraints	This package contains compartment constraint items.
glotaran.builtin.models.kinetic_spectrum.spectral_irf	
glotaran.builtin.models.kinetic_spectrum.spectral_matrix	Glotaran Spectral Matrix
glotaran.builtin.models.kinetic_spectrum.spectral_penalties	This package contains compartment constraint items.
glotaran.builtin.models.kinetic_spectrum.spectral_relations	Glotaran Spectral Relation
glotaran.builtin.models.kinetic_spectrum.spectral_shape	This package contains the spectral shape item.

kinetic_spectrum_dataset_descriptor

Classes

Summary

KineticSpectrumDatasetDescriptor

KineticSpectrumDatasetDescriptor

class glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.**Kin**

Bases: *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor*

Attributes Summary

baseline
initial_concentration
irf
label
megacomplex
scale
shape

baseline

KineticSpectrumDatasetDescriptor.**baseline**

initial_concentration

KineticSpectrumDatasetDescriptor.**initial_concentration**

irf

KineticSpectrumDatasetDescriptor.**irf**

label

KineticSpectrumDatasetDescriptor.**label**

megacomplex

KineticSpectrumDatasetDescriptor.**megacomplex**

scale

KineticSpectrumDatasetDescriptor.**scale**

shape

KineticSpectrumDatasetDescriptor.**shape**

Methods Summary

compartments	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
get_k_matrices	
mprint	

continues on next page

Table 68 – continued from previous page

validate

compartments

KineticSpectrumDatasetDescriptor.**compartments**()

fill

KineticSpectrumDatasetDescriptor.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KineticSpectrumDatasetDescriptor.**from_dict**(*values: dict*) → cls

from_list

classmethod KineticSpectrumDatasetDescriptor.**from_list**(*values: list*) → cls

get_k_matrices

KineticSpectrumDatasetDescriptor.**get_k_matrices**()

mprint

KineticSpectrumDatasetDescriptor.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

KineticSpectrumDatasetDescriptor.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property baseline

compartments()

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

get_k_matrices()

property initial_concentration

property irf

property label

property megacomplex

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale

property shape

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_spectrum_matrix

Glotaran kinetic spectrum Matrix

kinetic_spectrum_model

Functions

Summary

apply_kinetic_model_constraints
apply_spectral_penalties

continues on next page

Table 69 – continued from previous page

grouped	
has_kinetic_model_constraints	
has_spectral_penalties	
index_dependent	
kinetic_spectrum_matrix	
retrieve_spectral_clps	
spectral_matrix	Calculates the matrix.

apply_kinetic_model_constraints

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**apply_kinetic_model_co**

apply_spectral_penalties

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**apply_spectral_penalti**

grouped

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**grouped**(*model:*
glotaran.builtin.mode

has_kinetic_model_constraints

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**has_kinetic_model_cons**

has_spectral_penalties

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**has_spectral_penalties**

index_dependent

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**index_dependent**(*model:*
glotaran.b
→
bool

kinetic_spectrum_matrix

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**kinetic_spectrum_matri**

retrieve_spectral_clps

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**retrieve_spectral_clps**

spectral_matrix

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**spectral_matrix**(*dataset*, *axis*)

Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

Classes

Summary

KineticSpectrumModel

KineticSpectrumModel

class glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**KineticSpectrumM**

Bases: *glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel*

Attributes Summary

dataset	
equal_area_penalties	
global_dimension	
index_dependent_matrix	
initial_concentration	
irf	
k_matrix	
megacomplex	
model_dimension	
model_type	The type of the model as human readable string.
shape	
spectral_constraints	
spectral_relations	
weights	

dataset

KineticSpectrumModel.**dataset**

equal_area_penalties

KineticSpectrumModel.**equal_area_penalties**

global_dimension

KineticSpectrumModel.**global_dimension = 'spectral'**

index_dependent_matrix

KineticSpectrumModel.**index_dependent_matrix**

initial_concentration

KineticSpectrumModel.**initial_concentration**

irf

KineticSpectrumModel.**irf**

k_matrix

KineticSpectrumModel.**k_matrix**

megacomplex

KineticSpectrumModel.**megacomplex**

model_dimension

KineticSpectrumModel.**model_dimension = 'time'**

model_type

KineticSpectrumModel.**model_type**
The type of the model as human readable string.

shape

KineticSpectrumModel.**shape**

spectral_constraints

KineticSpectrumModel.**spectral_constraints**

spectral_relations

KineticSpectrumModel.**spectral_relations**

weights

KineticSpectrumModel.**weights**

Methods Summary

add_equal_area_penalties	
add_spectral_constraints	
add_spectral_relations	
add_weights	
additional_penalty_function	
constrain_matrix_function	
finalize_data	
from_dict	Creates a model from a dictionary.
get_dataset	
get_initial_concentration	
get_irf	
get_k_matrix	

continues on next page

Table 72 – continued from previous page

get_megacomplex	
get_shape	
global_matrix	Calculates the matrix.
grouped	
has_additional_penalty_function	
has_matrix_constraints_function	
index_dependent	
markdown	Formats the model as Markdown string.
matrix	
problem_list	Returns a list with all problems in the model and missing parameters if specified.
retrieve_clp_function	
set_dataset	
set_initial_concentration	
set_irf	
set_k_matrix	
set_megacomplex	
set_shape	
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

add_equal_area_penalties

KineticSpectrumModel.**add_equal_area_penalties**(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_pena

add_spectral_constraints

KineticSpectrumModel.**add_spectral_constraints**(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_cons

add_spectral_relations

KineticSpectrumModel.**add_spectral_relations**(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_relation

add_weights

KineticSpectrumModel.**add_weights**(*item:* glotaran.model.weight.Weight)

additional_penalty_function

KineticSpectrumModel.**additional_penalty_function**(*parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *clps: dict[str, list[np.ndarray]]*, *matrices: dict[str, np.ndarray | list[np.ndarray]]*, *data: dict[str, xr.Dataset]*, *group_tolerance: float*) → np.ndarray

constrain_matrix_function

KineticSpectrumModel.**constrain_matrix_function**(*dataset: str*, *parameters: ParameterGroup*, *clp_labels: list[str]*, *matrix: np.ndarray*, *index: float*) → tuple[list[str], np.ndarray]

finalize_data

KineticSpectrumModel.**finalize_data**(*problem: Problem*, *data: dict[str, xr.Dataset]*)

from_dict

classmethod KineticSpectrumModel.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset

KineticSpectrumModel.**get_dataset**(*label: str*) → *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descr*

get_initial_concentration

KineticSpectrumModel.**get_initial_concentration**(*label: str*) → *glotaran.builtin.models.kinetic_image.initial_concen*

get_irf

KineticSpectrumModel.**get_irf**(*label: str*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix

KineticSpectrumModel.**get_k_matrix**(*label: str*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex

KineticSpectrumModel.**get_megacomplex**(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.*

get_shape

KineticSpectrumModel.**get_shape**(*label: str*) → *glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape*

global_matrix

static KineticSpectrumModel.**global_matrix**(*dataset*, *axis*)

Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

grouped

KineticSpectrumModel.**grouped**()

has_additional_penalty_function

KineticSpectrumModel.**has_additional_penalty_function**() → bool

has_matrix_constraints_function

KineticSpectrumModel.**has_matrix_constraints_function**() → bool

index_dependent

KineticSpectrumModel.**index_dependent**() → bool

markdown

KineticSpectrumModel.**markdown**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*, *initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** – Parameter to include.
- **initial** – Initial values for the parameters.

matrix

static KineticSpectrumModel.**matrix**(*dataset_descriptor=None*, *axis=None*, *index=None*, *irf=None*)

problem_list

KineticSpectrumModel.**problem_list**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

retrieve_clp_function

KineticSpectrumModel.**retrieve_clp_function**(*parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *reduced_clp_labels: dict[str, list[str] | list[list[str]]]*, *reduced_clps: dict[str, list[np.ndarray]]*, *data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

set_dataset

KineticSpectrumModel.**set_dataset**(*label: str*, *item:* glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descri

set_initial_concentration

KineticSpectrumModel.**set_initial_concentration**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.initial_concent

set_irf

KineticSpectrumModel.**set_irf**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix

KineticSpectrumModel.**set_k_matrix**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex

KineticSpectrumModel.**set_megacomplex**(*label: str*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.

set_shape

KineticSpectrumModel.**set_shape**(*label: str*, *item:* glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape)

simulate

KineticSpectrumModel.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

KineticSpectrumModel.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate

KineticSpectrumModel.**validate**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

Methods Documentation

add_equal_area_penalties(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty)

add_spectral_constraints(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint

add_spectral_relations(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelation)

add_weights(*item:* glotaran.model.weight.Weight)

additional_penalty_function(*parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]]], clps: dict[str, list[np.ndarray]], matrices: dict[str, np.ndarray | list[np.ndarray]], data: dict[str, xr.Dataset], group_tolerance: float*) → np.ndarray

constrain_matrix_function(*dataset: str, parameters: ParameterGroup, clp_labels: list[str], matrix: np.ndarray, index: float*) → tuple[list[str], np.ndarray]

property dataset

property equal_area_penalties

finalize_data(*problem: Problem, data: dict[str, xr.Dataset]*)

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset(*label: str*) → *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpect*

get_initial_concentration(*label: str*) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialCon*

get_irf(*label: str*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix(*label: str*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex(*label: str*) → *glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.KineticImageMeg*

get_shape(*label: str*) → *glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape*

global_dimension = 'spectral'

static global_matrix(*dataset, axis*)

Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

grouped()

has_additional_penalty_function() → bool

has_matrix_constraints_function() → bool

index_dependent() → bool

property index_dependent_matrix

property initial_concentration

property irf

property k_matrix

markdown(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*, *initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** – Parameter to include.
- **initial** – Initial values for the parameters.

static matrix(*dataset_descriptor=None*, *axis=None*, *index=None*, *irf=None*)

property megacomplex

model_dimension = 'time'

property model_type

The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

retrieve_clp_function(*parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *reduced_clp_labels: dict[str, list[str] | list[list[str]]]*, *reduced_clps: dict[str, list[np.ndarray]]*, *data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

set_dataset(*label: str*, *item:* glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpe

set_initial_concentration(*label: str*, *item:* glotaran.builtin.models.kinetic_image.initial_concentration.InitialCo

set_irf(*label: str*, *item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix(*label: str*, *item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex(*label: str*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.KineticImageM

set_shape(*label: str*, *item:* glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape)

property shape

simulate(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.

- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

property spectral_constraints

property spectral_relations

valid(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property weights

kinetic_spectrum_result

Functions

Summary

finalize_kinetic_spectrum_result

finalize_kinetic_spectrum_result

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result.**finalize_kinetic_spec**

spectral_constraints

This package contains compartment constraint items.

Functions

Summary

apply_spectral_constraints

apply_spectral_constraints

glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**apply_spectral_constrain**

Classes

Summary

OnlyConstraint	A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.
SpectralConstraint	A compartment constraint is applied on one compartment on one or many intervals on the estimated axis type.
ZeroConstraint	A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

OnlyConstraint

class glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**OnlyConstraint**
Bases: object

A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.

Attributes Summary

compartment	
interval	
type	

compartment

OnlyConstraint.**compartment**

interval

OnlyConstraint.**interval**

type

OnlyConstraint.**type**

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

OnlyConstraint.**applies**(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

fill

OnlyConstraint.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod OnlyConstraint.**from_dict**(*values: dict*) → cls

from_list

classmethod OnlyConstraint.**from_list**(*values: list*) → cls

mprint

OnlyConstraint.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

OnlyConstraint.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

property compartment

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.

- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type

validate(*model: Model*, *parameters=None*) → list[str]

SpectralConstraint

class glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**SpectralConstraint**

Bases: object

A compartment constraint is applied on one compartment on one or many intervals on the estimated axis type.

There are three types: zero, equal and equal area. See the documentation of the respective classes for details.

Methods Summary

add_type

add_type

classmethod SpectralConstraint.**add_type**(*type_name: str*, *attribute_type: type*)

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

ZeroConstraint

class glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**ZeroConstraint**

Bases: object

A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

Attributes Summary

compartment	
interval	
type	

compartment

ZeroConstraint.**compartment**

interval

ZeroConstraint.**interval**

type

ZeroConstraint.**type**

Methods Summary

applies	Returns true if the indexx is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

ZeroConstraint.**applies**(*index: Any*) → bool
Returns true if the indexx is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

fill

ZeroConstraint.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod ZeroConstraint.**from_dict**(*values: dict*) → cls

from_list

classmethod ZeroConstraint.**from_list**(*values: list*) → cls

mprint

ZeroConstraint.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

ZeroConstraint.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool
Returns true if the indexx is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

property compartment

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.

- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type

validate(*model: Model*, *parameters=None*) → list[str]

spectral_irf

Classes

Summary

IrfGaussianCoherentArtifact	
IrfSpectralGaussian	
IrfSpectralMultiGaussian	Represents a gaussian IRF.

IrfGaussianCoherentArtifact

class glotaran.builtin.models.kinetic_spectrum.spectral_irf.**IrfGaussianCoherentArtifac**

Bases: *glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian*

Attributes Summary

backsweep	
backsweep_period	
center	
center_dispersion	
coherent_artifact_order	
coherent_artifact_width	
dispersion_center	
label	

continues on next page

Table 82 – continued from previous page

model_dispersion_with_wavenumber
normalize
scale
type
width
width_dispersion

backsweep

IrfGaussianCoherentArtifact.**backsweep**

backsweep_period

IrfGaussianCoherentArtifact.**backsweep_period**

center

IrfGaussianCoherentArtifact.**center**

center_dispersion

IrfGaussianCoherentArtifact.**center_dispersion**

coherent_artifact_order

IrfGaussianCoherentArtifact.**coherent_artifact_order**

coherent_artifact_width

IrfGaussianCoherentArtifact.**coherent_artifact_width**

dispersion_center

IrfGaussianCoherentArtifact.**dispersion_center**

label

IrfGaussianCoherentArtifact.**label**

model_dispersion_with_wavenumber

IrfGaussianCoherentArtifact.**model_dispersion_with_wavenumber**

normalize

IrfGaussianCoherentArtifact.**normalize**

scale

IrfGaussianCoherentArtifact.**scale**

type

IrfGaussianCoherentArtifact.**type**

width

IrfGaussianCoherentArtifact.**width**

width_dispersion

IrfGaussianCoherentArtifact.**width_dispersion**

Methods Summary

calculate
calculate_coherent_artifact
calculate_dispersion
clp_labels

continues on next page

Table 83 – continued from previous page

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfGaussianCoherentArtifact.**calculate**(*index*, *axis*)

calculate_coherent_artifact

IrfGaussianCoherentArtifact.**calculate_coherent_artifact**(*axis*)

calculate_dispersion

IrfGaussianCoherentArtifact.**calculate_dispersion**(*axis*)

clp_labels

IrfGaussianCoherentArtifact.**clp_labels**()

fill

IrfGaussianCoherentArtifact.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfGaussianCoherentArtifact.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfGaussianCoherentArtifact.**from_list**(*values: list*) → cls

mprint

IrfGaussianCoherentArtifact.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfGaussianCoherentArtifact.**parameter**(*index*)

validate

IrfGaussianCoherentArtifact.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep

property backsweep_period

calculate(*index*, *axis*)

calculate_coherent_artifact(*axis*)

calculate_dispersion(*axis*)

property center

property center_dispersion

clp_labels()

property coherent_artifact_order

property coherent_artifact_width

property dispersion_center

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

property model_dispersion_with_wavenumber

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize

parameter(*index*)

property scale

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

property width_dispersion

IrfSpectralGaussian

class glotaran.builtin.models.kinetic_spectrum.spectral_irf.**IrfSpectralGaussian**
Bases: *glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian*

Attributes Summary

backsweep
backsweep_period
center
center_dispersion
dispersion_center
label
model_dispersion_with_wavenumber
normalize
scale
type
width
width_dispersion

backsweep

IrfSpectralGaussian.**backsweep**

backsweep_period

IrfSpectralGaussian.**backsweep_period**

center

IrfSpectralGaussian.**center**

center_dispersion

IrfSpectralGaussian.**center_dispersion**

dispersion_center

IrfSpectralGaussian.**dispersion_center**

label

IrfSpectralGaussian.**label**

model_dispersion_with_wavenumber

IrfSpectralGaussian.**model_dispersion_with_wavenumber**

normalize

IrfSpectralGaussian.**normalize**

scale

IrfSpectralGaussian.**scale**

type

IrfSpectralGaussian.**type**

width

IrfSpectralGaussian.**width**

width_dispersion

IrfSpectralGaussian.**width_dispersion**

Methods Summary

calculate	
calculate_dispersion	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfSpectralGaussian.**calculate**(*index*, *axis*)

calculate_dispersion

IrfSpectralGaussian.**calculate_dispersion**(*axis*)

fill

IrfSpectralGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfSpectralGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfSpectralGaussian.**from_list**(*values: list*) → cls

mprint

IrfSpectralGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfSpectralGaussian.**parameter**(*index*)

validate

IrfSpectralGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep

property backsweep_period

calculate(*index*, *axis*)

calculate_dispersion(*axis*)

property center

property center_dispersion

property dispersion_center

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

property model_dispersion_with_wavenumber

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize

parameter(*index*)

property scale

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

property width_dispersion

IrfSpectralMultiGaussian

class glotaran.builtin.models.kinetic_spectrum.spectral_irf.**IrfSpectralMultiGaussian**
Bases: *glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian*

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Attributes Summary

backsweep
backsweep_period
center
center_dispersion
dispersion_center
label
model_dispersion_with_wavenumber
normalize
scale
type
width
width_dispersion

backsweep

IrfSpectralMultiGaussian.**backsweep**

backsweep_period

IrfSpectralMultiGaussian.**backsweep_period**

center

IrfSpectralMultiGaussian.**center**

center_dispersion

IrfSpectralMultiGaussian.**center_dispersion**

dispersion_center

IrfSpectralMultiGaussian.**dispersion_center**

label

IrfSpectralMultiGaussian.**label**

model_dispersion_with_wavenumber

IrfSpectralMultiGaussian.**model_dispersion_with_wavenumber**

normalize

IrfSpectralMultiGaussian.**normalize**

scale

IrfSpectralMultiGaussian.**scale**

type

IrfSpectralMultiGaussian.**type**

width

IrfSpectralMultiGaussian.**width**

width_dispersion

IrfSpectralMultiGaussian.**width_dispersion**

Methods Summary

calculate	
calculate_dispersion	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	

continues on next page

Table 87 – continued from previous page

from_list
mprint
parameter
validate

calculate

IrfSpectralMultiGaussian.**calculate**(*index*, *axis*)

calculate_dispersion

IrfSpectralMultiGaussian.**calculate_dispersion**(*axis*)

fill

IrfSpectralMultiGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfSpectralMultiGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfSpectralMultiGaussian.**from_list**(*values: list*) → cls

mprint

IrfSpectralMultiGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfSpectralMultiGaussian.**parameter**(*index*)

validate

IrfSpectralMultiGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep

property backsweep_period

calculate(*index*, *axis*)

calculate_dispersion(*axis*)

property center

property center_dispersion

property dispersion_center

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

property model_dispersion_with_wavenumber

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize

parameter(*index*)

property scale

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

property width_dispersion

spectral_matrix

Glotaran Spectral Matrix

spectral_penalties

This package contains compartment constraint items.

Classes

Summary

EqualAreaPenalty	An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual.

EqualAreaPenalty

class glotaran.builtin.models.kinetic_spectrum.spectral_penalties.**EqualAreaPenalty**
Bases: object

An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual. The additional residual is scaled with the weight.

Attributes Summary

parameter
source
source_intervals
target
target_intervals
weight

parameter

EqualAreaPenalty.**parameter**

source

EqualAreaPenalty.**source**

source_intervals

EqualAreaPenalty.**source_intervals**

target

EqualAreaPenalty.**target**

target_intervals

EqualAreaPenalty.**target_intervals**

weight

EqualAreaPenalty.**weight**

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

EqualAreaPenalty.**applies**(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

fill

EqualAreaPenalty.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod EqualAreaPenalty.**from_dict**(*values: dict*) → cls

from_list

classmethod EqualAreaPenalty.**from_list**(*values: list*) → cls

mprint

EqualAreaPenalty.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

EqualAreaPenalty.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property parameter

property source

property source_intervals

property target

property target_intervals

validate(*model: Model*, *parameters=None*) → list[str]

property weight

spectral_relations

Glotaran Spectral Relation

Functions

Summary

apply_spectral_relations
create_spectral_relation_matrix
retrieve_related_clps

apply_spectral_relations

glotaran.builtin.models.kinetic_spectrum.spectral_relations.**apply_spectral_relations**(*m*
K
ne
ic
Sp
tr
M
da
st
pa
ra
e-
te
Pa
ra
e-
te
G
cl
lis
m
tr
np
in
de
flo
—
tu
pl
np

create_spectral_relation_matrix

glotaran.builtin.models.kinetic_spectrum.spectral_relations.**create_spectral_relation_ma**

retrieve_related_clps

glotaran.builtin.models.kinetic_spectrum.spectral_relations.**retrieve_related_clps**(*model*
Ki-
net-
ic-
Spec-
trum-
Model
pa-
ram-
e-
ters:
Pa-
ram-
e-
ter-
Group
clp_la
dict[st
list[str
|
list[lis
clps:
dict[st
list[np
data:
dict[st
xr.Dat
→
dict[st
list[np

Classes

Summary

SpectralRelation

SpectralRelation

class glotaran.builtin.models.kinetic_spectrum.spectral_relations.**SpectralRelation**
Bases: object

Attributes Summary

compartment
interval
parameter
target

compartment

SpectralRelation.**compartment**

interval

SpectralRelation.**interval**

parameter

SpectralRelation.**parameter**

target

SpectralRelation.**target**

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	

continues on next page

Table 94 – continued from previous page

mprint
validate

applies

SpectralRelation.**applies**(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –
Returns applies
Return type bool

fill

SpectralRelation.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralRelation.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralRelation.**from_list**(*values: list*) → cls

mprint

SpectralRelation.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralRelation.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns applies
Return type bool

property compartment

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property parameter

property target

validate(*model: Model*, *parameters=None*) → list[str]

spectral_shape

This package contains the spectral shape item.

Classes

Summary

SpectralShape	Base class for spectral shapes
SpectralShapeGaussian	A gaussian spectral shape
SpectralShapeOne	A gaussian spectral shape
SpectralShapeZero	A gaussian spectral shape

SpectralShape

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShape**
Bases: object

Base class for spectral shapes

Methods Summary

add_type

add_type

classmethod SpectralShape.**add_type**(*type_name: str*, *attribute_type: type*)

Methods Documentation

classmethod **add_type**(*type_name: str*, *attribute_type: type*)

SpectralShapeGaussian

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeGaussian**
Bases: object

A gaussian spectral shape

Attributes Summary

amplitude
label
location
type
width

amplitude

SpectralShapeGaussian.**amplitude**

label

SpectralShapeGaussian.**label**

location

SpectralShapeGaussian.**location**

type

SpectralShapeGaussian.**type**

width

SpectralShapeGaussian.**width**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

SpectralShapeGaussian.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.
Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.
Returns shape
Return type numpy.ndarray

fill

SpectralShapeGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeGaussian.**from_list**(*values: list*) → cls

mprint

SpectralShapeGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property amplitude

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.
Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.
Returns shape
Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.

- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

property location

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type

validate(*model: Model*, *parameters=None*) → list[str]

property width

SpectralShapeOne

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeOne**

Bases: object

A gaussian spectral shape

Attributes Summary

label	
type	

label

SpectralShapeOne.**label**

type

SpectralShapeOne.**type**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	

continues on next page

Table 100 – continued from previous page

mprint
validate

calculate

SpectralShapeOne.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.

Returns **shape**

Return type numpy.ndarray

fill

SpectralShapeOne.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeOne.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeOne.**from_list**(*values: list*) → cls

mprint

SpectralShapeOne.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeOne.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.
Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.
Returns shape
Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type

validate(*model: Model*, *parameters=None*) → list[str]

SpectralShapeZero

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeZero**
Bases: object

A gaussian spectral shape

Attributes Summary

label
type

label

SpectralShapeZero.**label**

type

SpectralShapeZero.**type**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

SpectralShapeZero.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Only works after calling calling 'fill'.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.
Returns **shape**
Return type numpy.ndarray

fill

SpectralShapeZero.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeZero.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeZero.**from_list**(*values: list*) → cls

mprint

SpectralShapeZero.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeZero.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.

Only works after calling calling 'fill'.
Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.
Returns shape
Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod **from_list**(*values: list*) → cls

property **label**

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property **type**

validate(*model: Model*, *parameters=None*) → list[str]

12.1.3 cli

Modules

glotaran.cli.commands

glotaran.cli.main

commands

Modules

glotaran.cli.commands.explore
glotaran.cli.commands.export
glotaran.cli.commands.optimize
glotaran.cli.commands.pluginlist
glotaran.cli.commands.print
glotaran.cli.commands.util
glotaran.cli.commands.validate

explore

Functions

Summary

export	Exports data from netCDF4 to ascii.

export

glotaran.cli.commands.explore.**export** (*filename: str*, *select*, *out: str*, *name: str*)
Exports data from netCDF4 to ascii.

export

optimize

Functions

Summary

optimize_cmd	Optimizes a model.

optimize_cmd

glotaran.cli.commands.optimize.**optimize_cmd**(*dataformat: str*, *data: List[str]*, *out: str*, *nfev: int*, *nnls: bool*, *yes: bool*, *parameters_file: str*, *model_file: str*, *scheme_file: str*)

Optimizes a model. e.g.: glotaran optimize –

pluginlist

Functions

Summary

plugin_list_cmd	Prints a list of installed plugins.

plugin_list_cmd

glotaran.cli.commands.pluginlist.**plugin_list_cmd**()

Prints a list of installed plugins.

print

Functions

Summary

print_cmd	Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

print_cmd

glotaran.cli.commands.print.**print_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

util

Functions

Summary

load_dataset_file	

continues on next page

Table 109 – continued from previous page

load_model_file
load_parameter_file
load_scheme_file
select_data
select_name
signature_analysis
write_data

load_dataset_file

glotaran.cli.commands.util.**load_dataset_file**(*filename*, *fmt=None*, *verbose=False*)

load_model_file

glotaran.cli.commands.util.**load_model_file**(*filename*, *verbose=False*)

load_parameter_file

glotaran.cli.commands.util.**load_parameter_file**(*filename*, *fmt=None*, *verbose=False*)

load_scheme_file

glotaran.cli.commands.util.**load_scheme_file**(*filename*, *verbose=False*)

select_data

glotaran.cli.commands.util.**select_data**(*data*, *dim*, *selection*)

select_name

glotaran.cli.commands.util.**select_name**(*filename*, *dataset*)

signature_analysis

glotaran.cli.commands.util.**signature_analysis**(*cmd*)

write_data

glotaran.cli.commands.util.**write_data**(*data*, *out*)

Classes

Summary

ValOrRangeOrList	

ValOrRangeOrList

class glotaran.cli.commands.util.**ValOrRangeOrList**
Bases: click.types.ParamType

Attributes Summary

envvar_list_splitter	if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up.
is_composite	
name	the descriptive name of this type

envvar_list_splitter

ValOrRangeOrList.**envvar_list_splitter = None**
if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by os.path.pathsep by default (":" on Unix and ";" on Windows).

is_composite

ValOrRangeOrList.**is_composite = False**

name

ValOrRangeOrList.**name = 'number or range or list'**
the descriptive name of this type

Methods Summary

convert	Converts the value.
fail	Helper method to fail with an invalid value message.
get_metavar	Returns the metavar default for this param if it provides one.
get_missing_message	Optionally might return extra information about a missing parameter.
split_envvar_value	Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

convert

ValOrRangeOrList.**convert**(*value*, *param*, *ctx*)
Converts the value. This is not invoked for values that are *None* (the missing value).

fail

ValOrRangeOrList.**fail**(*message*, *param=None*, *ctx=None*)
Helper method to fail with an invalid value message.

get_metavar

ValOrRangeOrList.**get_metavar**(*param*)
Returns the metavar default for this param if it provides one.

get_missing_message

ValOrRangeOrList.**get_missing_message**(*param*)
Optionally might return extra information about a missing parameter.

New in version 2.0.

split_envvar_value

ValOrRangeOrList.**split_envvar_value**(*rv*)
Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

Methods Documentation

convert(*value*, *param*, *ctx*)
Converts the value. This is not invoked for values that are *None* (the missing value).

envvar_list_splitter = None
if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by `os.path.pathsep` by default (":" on Unix and ";" on Windows).

fail(*message*, *param=None*, *ctx=None*)
Helper method to fail with an invalid value message.

get_metavar(*param*)
Returns the metavar default for this param if it provides one.

get_missing_message(*param*)
Optionally might return extra information about a missing parameter.

New in version 2.0.

is_composite = False

name = 'number or range or list'
the descriptive name of this type

split_envvar_value(*rv*)
Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

validate

Functions

Summary

validate_cmd	Validates a model file and optionally a parameter file.

validate_cmd

glotaran.cli.commands.validate.**validate_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Validates a model file and optionally a parameter file.

main

Classes

Summary

Cli

Cli

class glotaran.cli.main.**Cli**(**args*, ***kwargs*)

Bases: click.core.Group

Attributes Summary

allow_extra_args	the default for the Context.allow_extra_args flag.
allow_interspersed_args	the default for the Context.allow_interspersed_args flag.
ignore_unknown_options	the default for the Context.ignore_unknown_options flag.

allow_extra_args

Cli.**allow_extra_args = True**

the default for the Context.allow_extra_args flag.

allow_interspersed_args

Cli.**allow_interspersed_args = False**
the default for the Context.allow_interspersed_args flag.

ignore_unknown_options

Cli.**ignore_unknown_options = False**
the default for the Context.ignore_unknown_options flag.

Methods Summary

add_command	Registers another Command with this group.
collect_usage_pieces	Returns all the pieces that go into the usage line and returns it as a list of strings.
command	Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.
format_commands	Extra format methods for multi methods that adds all the commands after the options.
format_epilog	Writes the epilog into the formatter if it exists.
format_help	Writes the help into the formatter if it exists.
format_help_text	Writes the help text to the formatter if it exists.
format_options	Writes all the options into the formatter if they exist.
format_usage	Writes the usage line into the formatter.
get_command	Given a context and a command name, this returns a Command object if it exists or returns *None*.
get_help	Formats the help into a string and returns it.
get_help_option	Returns the help option object.
get_help_option_names	Returns the names for the help option.
get_params	
get_short_help_str	Gets short help for the command or makes it by shortening the long help string.
get_usage	Formats the usage line into a string and returns it.
group	A shortcut decorator for declaring and attaching a group to the group.
invoke	Given a context, this invokes the attached callback (if it exists) in the right way.
list_commands	Returns a list of subcommand names in the order they should appear.
list_commands_for_help	reorder the list of commands when listing the help
main	This is the way to invoke a script with all the bells and whistles as a command line application.

continues on next page

Table 116 – continued from previous page

make_context	This function when given an info name and arguments will kick off the parsing and create a new `Context`.
make_parser	Creates the underlying option parser for this command.
parse_args	Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary.
resolve_command	
resultcallback	Adds a result callback to the chain command.

add_command

`Cli.`**`add_command`**(*cmd*, *name=None*)

Registers another `Command` with this group. If the name is not provided, the name of the command is used.

collect_usage_pieces

`Cli.`**`collect_usage_pieces`**(*ctx*)

Returns all the pieces that go into the usage line and returns it as a list of strings.

command

`Cli.`**`command`**(**args*, ***kwargs*)

Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.

format_commands

`Cli.`**`format_commands`**(*ctx*, *formatter*)

Extra format methods for multi methods that adds all the commands after the options.

format_epilog

`Cli.`**`format_epilog`**(*ctx*, *formatter*)

Writes the epilog into the formatter if it exists.

format_help

Cli.**format_help**(*ctx*, *formatter*)
Writes the help into the formatter if it exists.

This is a low-level method called by *get_help()*.

This calls the following methods:
- *format_usage()*
- *format_help_text()*
- *format_options()*
- *format_epilog()*

format_help_text

Cli.**format_help_text**(*ctx*, *formatter*)
Writes the help text to the formatter if it exists.

format_options

Cli.**format_options**(*ctx*, *formatter*)
Writes all the options into the formatter if they exist.

format_usage

Cli.**format_usage**(*ctx*, *formatter*)
Writes the usage line into the formatter.

This is a low-level method called by *get_usage()*.

get_command

Cli.**get_command**(*ctx*, *cmd_name*)
Given a context and a command name, this returns a Command object if it exists or returns *None*.

get_help

Cli.**get_help**(*ctx*)
Formats the help into a string and returns it.

Calls *format_help()* internally.

get_help_option

Cli.**get_help_option**(*ctx*)
Returns the help option object.

get_help_option_names

Cli.**get_help_option_names**(*ctx*)
Returns the names for the help option.

get_params

Cli.**get_params**(*ctx*)

get_short_help_str

Cli.**get_short_help_str**(*limit=45*)
Gets short help for the command or makes it by shortening the long help string.

get_usage

Cli.**get_usage**(*ctx*)
Formats the usage line into a string and returns it.

Calls *format_usage()* internally.

group

Cli.**group**(**args*, ***kwargs*)
A shortcut decorator for declaring and attaching a group to the group. This takes the same arguments as *group()* but immediately registers the created command with this instance by calling into *add_command()*.

invoke

Cli.**invoke**(*ctx*)
Given a context, this invokes the attached callback (if it exists) in the right way.

list_commands

Cli.**list_commands**(*ctx*)
Returns a list of subcommand names in the order they should appear.

list_commands_for_help

Cli.**list_commands_for_help**(*ctx*)
reorder the list of commands when listing the help

main

Cli.**main**(*args=None*, *prog_name=None*, *complete_var=None*, *standalone_mode=True*, ***extra*)
This is the way to invoke a script with all the bells and whistles as a command line application. This will always terminate the application after a call. If this is not wanted, `SystemExit` needs to be caught.

This method is also available by directly calling the instance of a `Command`.

New in version 3.0: Added the *standalone_mode* flag to control the standalone mode.

Parameters

- **args** – the arguments that should be used for parsing. If not provided, `sys.argv[1:]` is used.
- **prog_name** – the program name that should be used. By default the program name is constructed by taking the file name from `sys.argv[0]`.
- **complete_var** – the environment variable that controls the bash completion support. The default is `"_<prog_name>_COMPLETE"` with prog_name in uppercase.
- **standalone_mode** – the default behavior is to invoke the script in standalone mode. Click will then handle exceptions and convert them into error messages and the function will never return but shut down the interpreter. If this is set to *False* they will be propagated to the caller and the return value of this function is the return value of *invoke()*.
- **extra** – extra keyword arguments are forwarded to the context constructor. See `Context` for more information.

make_context

Cli.**make_context**(*info_name*, *args*, *parent=None*, ***extra*)
This function when given an info name and arguments will kick off the parsing and create a new `Context`. It does not invoke the actual command callback though.

Parameters

- **info_name** – the info name for this invokation. Generally this is the most descriptive name for the script or command. For the toplevel script it's usually the name of the script, for commands below it it's the name of the script.
- **args** – the arguments to parse as list of strings.
- **parent** – the parent context if available.
- **extra** – extra keyword arguments forwarded to the context constructor.

make_parser

Cli.**make_parser**(*ctx*)
Creates the underlying option parser for this command.

parse_args

Cli.**parse_args**(*ctx*, *args*)
Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary. This is automatically invoked by *make_context()*.

resolve_command

Cli.**resolve_command**(*ctx*, *args*)

resultcallback

Cli.**resultcallback**(*replace=False*)
Adds a result callback to the chain command. By default if a result callback is already registered this will chain them but this can be disabled with the *replace* parameter. The result callback is invoked with the return value of the subcommand (or the list of return values from all subcommands if chaining is enabled) as well as the parameters as they would be passed to the main callback.

Example:

```
@click.group()
@click.option('-i', '--input', default=23)
def cli(input):
    return 42

@cli.resultcallback()
def process_result(result, input):
    return result + input
```

New in version 3.0.
Parameters **replace** – if set to *True* an already existing result callback will be removed.

Methods Documentation

add_command(*cmd*, *name=None*)
Registers another Command with this group. If the name is not provided, the name of the command is used.

allow_extra_args = True
the default for the Context.allow_extra_args flag.

allow_interspersed_args = False
the default for the Context.allow_interspersed_args flag.

callback
the callback to execute when the command fires. This might be *None* in which case nothing happens.

collect_usage_pieces(*ctx*)
Returns all the pieces that go into the usage line and returns it as a list of strings.

command(**args*, ***kwargs*)
Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.

commands
the registered subcommands by their exported names.

context_settings
an optional dictionary with defaults passed to the context.

format_commands(*ctx*, *formatter*)
Extra format methods for multi methods that adds all the commands after the options.

format_epilog(*ctx*, *formatter*)
Writes the epilog into the formatter if it exists.

format_help(*ctx*, *formatter*)
Writes the help into the formatter if it exists.

This is a low-level method called by *get_help()*.

This calls the following methods:
- *format_usage()*
- *format_help_text()*
- *format_options()*
- *format_epilog()*

format_help_text(*ctx*, *formatter*)
Writes the help text to the formatter if it exists.

format_options(*ctx*, *formatter*)
Writes all the options into the formatter if they exist.

format_usage(*ctx*, *formatter*)
Writes the usage line into the formatter.

This is a low-level method called by *get_usage()*.

get_command(*ctx*, *cmd_name*)
Given a context and a command name, this returns a `Command` object if it exists or returns *None*.

get_help(*ctx*)
Formats the help into a string and returns it.

Calls *format_help()* internally.

get_help_option(*ctx*)
Returns the help option object.

get_help_option_names(*ctx*)
Returns the names for the help option.

get_params(*ctx*)

get_short_help_str(*limit=45*)
Gets short help for the command or makes it by shortening the long help string.

get_usage(*ctx*)
Formats the usage line into a string and returns it.

Calls *format_usage()* internally.

group(**args*, ***kwargs*)
A shortcut decorator for declaring and attaching a group to the group. This takes the same arguments as *group()* but immediately registers the created command with this instance by calling into *add_command()*.

ignore_unknown_options = False
the default for the `Context.ignore_unknown_options` flag.

invoke(*ctx*)
Given a context, this invokes the attached callback (if it exists) in the right way.

list_commands(*ctx*)
Returns a list of subcommand names in the order they should appear.

list_commands_for_help(*ctx*)
reorder the list of commands when listing the help

main(*args=None*, *prog_name=None*, *complete_var=None*, *standalone_mode=True*, ***extra*)
This is the way to invoke a script with all the bells and whistles as a command line application. This will always terminate the application after a call. If this is not wanted, `SystemExit` needs to be caught.

This method is also available by directly calling the instance of a `Command`.

New in version 3.0: Added the *standalone_mode* flag to control the standalone mode.

Parameters

- **args** – the arguments that should be used for parsing. If not provided, `sys.argv[1:]` is used.
- **prog_name** – the program name that should be used. By default the program name is constructed by taking the file name from `sys.argv[0]`.
- **complete_var** – the environment variable that controls the bash completion support. The default is `"_<prog_name>_COMPLETE"` with prog_name in uppercase.
- **standalone_mode** – the default behavior is to invoke the script in standalone mode. Click will then handle exceptions and convert them into error messages and the function will never return but shut down the interpreter. If this is set to *False* they will be propagated to the caller and the return value of this function is the return value of *invoke()*.
- **extra** – extra keyword arguments are forwarded to the context constructor. See `Context` for more information.

make_context(*info_name*, *args*, *parent=None*, ***extra*)
This function when given an info name and arguments will kick off the parsing and create a new `Context`. It does not invoke the actual command callback though.

Parameters

- **info_name** – the info name for this invokation. Generally this is the most descriptive name for the script or command. For the toplevel script it's usually the name of the script, for commands below it it's the name of the script.
- **args** – the arguments to parse as list of strings.
- **parent** – the parent context if available.
- **extra** – extra keyword arguments forwarded to the context constructor.

make_parser(*ctx*)
Creates the underlying option parser for this command.

name
the name the command thinks it has. Upon registering a command on a `Group` the group will default the command name with this information. You should instead use the `Context`'s `info_name` attribute.

params
the list of parameters for this command in the order they should show up in the help page and execute. Eager parameters will automatically be handled before non eager ones.

parse_args(*ctx*, *args*)
Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary. This is automatically invoked by *make_context()*.

resolve_command(*ctx*, *args*)

result_callback
The result callback that is stored. This can be set or overridden with the *resultcallback()* decorator.

resultcallback(*replace=False*)
Adds a result callback to the chain command. By default if a result callback is already registered this will chain them but this can be disabled with the *replace* parameter. The result callback is invoked with the return value of the subcommand (or the list of return values from all subcommands if chaining is enabled) as well as the parameters as they would be passed to the main callback.

Example:

```
@click.group()
@click.option('-i', '--input', default=23)
def cli(input):
    return 42

@cli.resultcallback()
def process_result(result, input):
    return result + input
```

New in version 3.0.
Parameters **replace** – if set to *True* an already existing result callback will be removed.

12.1.4 examples

Modules

glotaran.examples.sequential

sequential

12.1.5 io

Functions for data IO

Modules

glotaran.io.prepare_dataset
glotaran.io.reader

prepare_dataset

Functions

Summary

prepare_time_trace_dataset	Prepares a time trace for global analysis.

prepare_time_trace_dataset

glotaran.io.prepare_dataset.**prepare_time_trace_dataset**(*dataset: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *weight: Optional[numpy.ndarray] = None*, *irf: Optional[Union[numpy.ndarray, xarray.core.dataarray.DataArray]] = None*) → xarray.core.dataset.Dataset

Prepares a time trace for global analysis.

Parameters

- **dataset** – The dataset.
- **weight** – A weight for the dataset.
- **irf** – An IRF for the dataset.

reader

Functions

Summary

file_reader
read_data_file

file_reader

glotaran.io.reader.**file_reader**(*extension: Optional[str] = None*, *name: Optional[str] = None*)

read_data_file

glotaran.io.reader.**read_data_file**(*filename: str*, *fmt: Optional[str] = None*) → xarray.core.dataset.Dataset

Classes

Summary

Reader

Reader

class glotaran.io.reader.**Reader**(*extension*, *name*, *reader*)

Bases: object

Methods Summary

—

Methods Documentation

12.1.6 model

Glotaran Model Package

This package contains the Glotaran's base model object, the model decorators and common model items.

Modules

glotaran.model.attribute	The model attribute decorator.
glotaran.model.base_model	A base class for global analysis models.
glotaran.model.dataset_descriptor	The DatasetDescriptor class.
glotaran.model.decorator	The model decorator.
glotaran.model.property	The model property class.
glotaran.model.util	Helper functions.
glotaran.model.weight	The Weight property class.

attribute

The model attribute decorator.

Functions

Summary

model_attribute	The *@model_attribute* decorator adds the given properties to the class.
model_attribute_typed	The model_attribute_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.

model_attribute

glotaran.model.attribute.**model_attribute**(*properties: Any | dict[str, dict[str, Any]] = {}*, *has_type: bool = False*, *no_label: bool = False*) → Callable

The *@model_attribute* decorator adds the given properties to the class. Further it adds classmethods for deserialization, validation and printing.

By default, a *label* property is added.

The *properties* dictionary contains the name of the properties as keys. The values must be either a *type* or dictionary with the following values:

- type: a *type* (required)
- doc: a string for documentation (optional)
- default: a default value (optional)

- allow_none: if *True*, the property can be set to None (optional)

Classes with the *model_attribute* decorator intended to be used in glotaran models.

Parameters

- **properties** – A dictionary of property names and options.
- **has_type** – If true, a type property will added. Used for model attributes, which can have more then one type.
- **no_label** – If true no label property will be added.

model_attribute_typed

glotaran.model.attribute.**model_attribute_typed**(*types: dict[str, Any]*, *no_label=False*)

The model_attribute_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.

Parameters

- **types** – A dictionary of types and options.
- **no_label** – If *True* no label property will be added.

base_model

A base class for global analysis models.

Classes

Summary

Model	A base class for global analysis models.

Model

class glotaran.model.base_model.**Model**

Bases: object

A base class for global analysis models.

Attributes Summary

index_dependent_matrix	
model_type	The type of the model as human readable string.

index_dependent_matrix

Model.**index_dependent_matrix**

model_type

Model.**model_type**
The type of the model as human readable string.

Methods Summary

from_dict	Creates a model from a dictionary.
markdown	Formats the model as Markdown string.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

from_dict

classmethod Model.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*
Creates a model from a dictionary.
Parameters **model_dict** – Dictionary containing the model.

markdown

Model.**markdown**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*, *initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str
Formats the model as Markdown string.

Parameters will be included if specified.
Parameters
- **parameter** – Parameter to include.
- **initial** – Initial values for the parameters.

problem_list

Model.**problem_list**(*parameters: ParameterGroup = None*) → list[str]
Returns a list with all problems in the model and missing parameters if specified.
Parameters **parameter** – The parameter to validate.

simulate

Model.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset
Simulates the model.
Parameters
- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

Model.**valid**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool
Returns *True* if the number problems in the model is 0, else *False*
Parameters **parameter** – The parameter to validate.

validate

Model.**validate**(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str
Returns a string listing all problems in the model and missing parameters if specified.
Parameters **parameter** – The parameter to validate.

Methods Documentation

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*
Creates a model from a dictionary.
Parameters **model_dict** – Dictionary containing the model.

property index_dependent_matrix

markdown(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*, *initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str
Formats the model as Markdown string.

Parameters will be included if specified.
Parameters
- **parameter** – Parameter to include.

- **initial** – Initial values for the parameters.

property model_type
The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]
Returns a list with all problems in the model and missing parameters if specified.
Parameters **parameter** – The parameter to validate.

simulate(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset
Simulates the model.
Parameters
- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool
Returns *True* if the number problems in the model is 0, else *False*
Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str
Returns a string listing all problems in the model and missing parameters if specified.
Parameters **parameter** – The parameter to validate.

dataset_descriptor

The DatasetDescriptor class.

Classes

Summary

DatasetDescriptor	A *DatasetDescriptor* describes a dataset in terms of a glotaran model.

DatasetDescriptor

class glotaran.model.dataset_descriptor.**DatasetDescriptor**
Bases: object

A *DatasetDescriptor* describes a dataset in terms of a glotaran model. It contains references to model items which describe the physical model for a given dataset.

A general dataset describtor assigns one or more megacomplexes and a scale parameter.

Attributes Summary

label
megacomplex
scale

label

DatasetDescriptor.**label**

megacomplex

DatasetDescriptor.**megacomplex**

scale

DatasetDescriptor.**scale**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

fill

DatasetDescriptor.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod DatasetDescriptor.**from_dict**(*values: dict*) → cls

from_list

classmethod DatasetDescriptor.**from_list**(*values: list*) → cls

mprint

DatasetDescriptor.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

DatasetDescriptor.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label

property megacomplex

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale

validate(*model: Model*, *parameters=None*) → list[str]

decorator

The model decorator.

Functions

Summary

model	The *@model* decorator is intended to be used on subclasses of `glotaran.model.Model`.

model

glotaran.model.decorator.**model**(*model_type: str, attributes: Optional[Dict[str, Any]] = None, dataset_type: Type[glotaran.model.dataset_descriptor.DatasetDescriptor] = <class 'glotaran.model.dataset_descriptor.DatasetDescriptor'>, megacomplex_type: Optional[Any] = None, matrix: Optional[Union[Callable[[Type[glotaran.model.dataset_descriptor.DatasetDescriptor], xarray.core.dataset.Dataset], Tuple[List[str], numpy.ndarray]], Callable[[Type[glotaran.model.dataset_descriptor.DatasetDescriptor], xarray.core.dataset.Dataset, Any], Tuple[List[str], numpy.ndarray]]]] = None, global_matrix: Optional[Callable[[Type[glotaran.model.dataset_descriptor.DatasetDescriptor], numpy.ndarray], Tuple[List[str], numpy.ndarray]]] = None, model_dimension: Optional[str] = None, global_dimension: Optional[str] = None, has_matrix_constraints_function: Optional[Callable[[Type[glotaran.model.base_model.Model]], bool]] = None, constrain_matrix_function: Optional[Callable[[Type[glotaran.model.base_model.Model], glotaran.parameter.parameter_group.ParameterGroup, List[str], numpy.ndarray, float], Tuple[List[str], numpy.ndarray]]] = None, retrieve_clp_function: Optional[Callable[[Type[glotaran.model.base_model.Model], glotaran.parameter.parameter_group.ParameterGroup, Dict[str, Union[List[str], List[List[str]]]], Dict[str, Union[List[str], List[List[str]]]], Dict[str, List[numpy.ndarray]], Dict[str, xarray.core.dataset.Dataset]], Dict[str, List[numpy.ndarray]]]] = None, has_additional_penalty_function: Optional[Callable[[Type[glotaran.model.base_model.Model]], bool]] = None, additional_penalty_function: Optional[Callable[[Type[glotaran.model.base_model.Model], glotaran.parameter.parameter_group.ParameterGroup, Dict[str, Union[List[str], List[List[str]]]], Dict[str, List[numpy.ndarray]], Dict[str, Union[numpy.ndarray, List[numpy.ndarray]]], Dict[str, xarray.core.dataset.Dataset], float], numpy.ndarray]] = None, finalize_data_function: Optional[Callable[[glotaran.analysis.problem.Problem, Dict[str, xarray.core.dataset.Dataset]], None]] = None, grouped: Union[bool, Callable[[Type[glotaran.model.base_model.Model]], bool]] = False, index_dependent: Union[bool, Callable[[Type[glotaran.model.base_model.Model]], bool]] = False*) → Callable

The @*model* decorator is intended to be used on subclasses of `glotaran.model.Model`. It creates properties for the given attributes as well as functions to add access them. Also it adds the functions (e.g. for *matrix*) to the model ensures they are added wrapped in a correct way.

Parameters

- **model_type** (*str*) – Human readable string used by the parser to identify the correct model.
- **attributes** (*Dict[str, Any], optional*) – A dictionary of attribute names and types. All types must be decorated with the `glotaran.model.model_attribute()` decorator, by default None.
- **dataset_type** (*Type[DatasetDescriptor], optional*) – A subclass of `DatasetDescriptor`, by default DatasetDescriptor
- **megacomplex_type** (*Any, optional*) – A class for the model megacomplexes. The class must be decorated with the `glotaran.model.model_attribute()` decorator, by default None
- **matrix** (*Union[MatrixFunction, IndexDependentMatrixFunction], optional*) – A function to calculate the matrix for the model, by default None
- **global_matrix** (*GlobalMatrixFunction, optional*) – A function to calculate the global matrix for the model, by default None
- **model_dimension** (*str, optional*) – The name of model matrix row dimension, by default None
- **global_dimension** (*str, optional*) – The name of model global matrix row dimension, by default None
- **has_matrix_constraints_function** (*Callable[[Type[Model]], bool], optional*) – True if the model as a constrain_matrix_function set, by default None
- **constrain_matrix_function** (*ConstrainMatrixFunction, optional*) – A function to constrain the global matrix for the model, by default None
- **retrieve_clp_function** (*RetrieveClpFunction, optional*) – A function to retrieve the full clp from the reduced, by default None
- **has_additional_penalty_function** (*Callable[[Type[Model]], bool], optional*) – True if model has a additional_penalty_function set, by default None
- **additional_penalty_function** (*PenaltyFunction, optional*) – A function to calculate additional penalties when optimizing the model, by default None
- **finalize_data_function** (*FinalizeFunction, optional*) – A function to finalize data after optimization, by default None
- **grouped** (*Union[bool, Callable[[Type[Model]], bool]], optional*) – True if model described a grouped problem, by default False
- **index_dependent** (*Union[bool, Callable[[Type[Model]], bool]], optional*) – True if model described a index dependent problem, by default False

Returns Returns a decorated model function

Return type Callable

Raises

- **ValueError** – If model implements meth:*has_matrix_constraints_function* but not meth:*constrain_matrix_function* and meth:*retrieve_clp_function*
- **ValueError** – If model implements meth:*has_additional_penalty_function* but not meth:*additional_penalty_function*

property

The model property class.

Classes

Summary

ModelProperty

ModelProperty

class glotaran.model.property.**ModelProperty**(*cls*, *name*, *prop_type*, *doc*, *default*, *allow_none*)

Bases: property

Attributes Summary

allow_none
fdel
fget
fset

allow_none

ModelProperty.**allow_none**

fdel

ModelProperty.**fdel**

fget

ModelProperty.**fget**

fset

ModelProperty.**fset**

Methods Summary

deleter	Descriptor to change the deleter on a property.
fill	
getter	Descriptor to change the getter on a property.
setter	Descriptor to change the setter on a property.
validate	

deleter

ModelProperty.**deleter**()
Descriptor to change the deleter on a property.

fill

ModelProperty.**fill**(*value*, *model*, *parameter*)

getter

ModelProperty.**getter**()
Descriptor to change the getter on a property.

setter

ModelProperty.**setter**()
Descriptor to change the setter on a property.

validate

ModelProperty.**validate**(*value*, *model*, *parameters=None*) → List[str]

Methods Documentation

property allow_none

deleter()
Descriptor to change the deleter on a property.

fdel

fget

fill(*value*, *model*, *parameter*)

fset

getter()
Descriptor to change the getter on a property.

setter()
Descriptor to change the setter on a property.

validate(*value*, *model*, *parameters=None*) → List[str]

util

Helper functions.

Functions

Summary

wrap_func_as_method	A decorator to wrap a function as class method.

wrap_func_as_method

glotaran.model.util.**wrap_func_as_method**(*cls: Any*, *name: Optional[str] = None*, *annotations: Optional[str] = None*, *doc: Optional[str] = None*) → Callable

A decorator to wrap a function as class method.

Notes

Only for internal use.

Parameters

- **cls** – The class in which the function will be wrapped.
- **name** – The name of method. If *None*, the original function's name is used.
- **annotations** – The annotations of the method. If *None*, the original function's annotations are used.
- **doc** – The documentation of the method. If *None*, the original function's documentation is used.

weight

The Weight property class.

Classes

Summary

Weight	The *Weight* class describes a value by which a dataset will scaled.

Weight

class glotaran.model.weight.**Weight**

Bases: object

The *Weight* class describes a value by which a dataset will scaled.

global_interval and *model_interval* are optional. The whole range of the dataset will be used if not set.

Attributes Summary

datasets
global_interval
model_interval
value

datasets

Weight.**datasets**

global_interval

Weight.**global_interval**

model_interval

Weight.**model_interval**

value

Weight.**value**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

fill

Weight.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod Weight.**from_dict**(*values: dict*) → cls

from_list

classmethod Weight.**from_list**(*values: list*) → cls

mprint

Weight.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

Weight.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property datasets

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property global_interval

property model_interval

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate(*model: Model*, *parameters=None*) → list[str]

property value

12.1.7 parameter

Modules

glotaran.parameter.parameter	The parameter class.
glotaran.parameter.parameter_group	The parameter group class

parameter

The parameter class.

Classes

Summary

Keys	Keys for parameter options.
Parameter	A parameter for optimization.

Keys

class glotaran.parameter.parameter.**Keys**

Bases: object

Keys for parameter options.

Attributes Summary

EXPR
MAX
MIN
NON_NEG
VARY

EXPR

Keys.**EXPR = 'expr'**

MAX

Keys.**MAX = 'max'**

MIN

Keys.**MIN = 'min'**

NON_NEG

Keys.**NON_NEG = 'non-negative'**

VARY

Keys.**VARY = 'vary'**

Methods Summary

—

Methods Documentation

EXPR = 'expr'

MAX = 'max'

MIN = 'min'

NON_NEG = 'non-negative'

VARY = 'vary'

Parameter

class glotaran.parameter.parameter.**Parameter**(*label: str = None*, *full_label: str = None*, *expression: str = None*, *maximum: int | float = inf*, *minimum: int | float = -inf*, *non_negative: bool = False*, *value: float = None*, *vary: bool = True*)

Bases: object

A parameter for optimization.

Parameters

- **label** – The label of the parameter.
- **full_label** (*str*) – The label of the parameter with its path in a parameter group prepended.

Attributes Summary

expression	The expression of the parameter.
full_label	The label of the parameter with its path in a parameter group prepended.
label	Label of the parameter
maximum	The upper bound of the parameter.
minimum	The lower bound of the parameter.
non_negative	Indicates if the parameter is non-negativ.
standard_error	The standard error of the optimized parameter.
transformed_expression	The expression of the parameter transformed for evaluation within a *ParameterGroup*.
value	The value of the parameter
vary	Indicates if the parameter should be optimized.

expression

Parameter.**expression**
The expression of the parameter.

full_label

Parameter.**full_label**
The label of the parameter with its path in a parameter group prepended.

label

Parameter.**label**
Label of the parameter

maximum

Parameter.**maximum**
The upper bound of the parameter.

minimum

Parameter.**minimum**
The lower bound of the parameter.

non_negative

Parameter.**non_negative**
Indicates if the parameter is non-negativ.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Always *False* if *expression* is not *None*.

standard_error

Parameter.**standard_error**
The standard error of the optimized parameter.

transformed_expression

Parameter.**transformed_expression**
The expression of the parameter transformed for evaluation within a *ParameterGroup*.

value

Parameter.**value**
The value of the parameter

vary

Parameter.**vary**
Indicates if the parameter should be optimized.

Always *False* if *expression* is not *None*.

Methods Summary

from_list_or_value	Creates a parameter from a list or numeric value.
get_value_and_bounds_for_optimization	Gets the parameter value and bounds with expression and non-negative constraints applied.
set_from_group	Sets all values of the parameter to the values of the corresponding parameter in the group.
set_value_from_optimization	Sets the value from an optimization result and reverses non-negative transformation.
valid_label	Returns true if the *label* is valid string.

from_list_or_value

classmethod Parameter.**from_list_or_value**(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → Parameter

Creates a parameter from a list or numeric value.

Parameters

- **value** – The list or numeric value.
- **default_options** – A dictionary of default options.
- **label** – The label of the parameter.

get_value_and_bounds_for_optimization

Parameter.**get_value_and_bounds_for_optimization**() → tuple[float, float, float]

Gets the parameter value and bounds with expression and non-negative constraints applied.

set_from_group

Parameter.**set_from_group**(*group: ParameterGroup*)

Sets all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** – The `glotaran.parameter.ParameterGroup`.

set_value_from_optimization

Parameter.**set_value_from_optimization**(*value: float*)

Sets the value from an optimization result and reverses non-negative transformation.

valid_label

classmethod Parameter.**valid_label**(*label: str*) → bool

Returns true if the *label* is valid string.

Methods Documentation

property expression

The expression of the parameter.

classmethod from_list_or_value(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → *Parameter*

Creates a parameter from a list or numeric value.

Parameters

- **value** – The list or numeric value.
- **default_options** – A dictionary of default options.
- **label** – The label of the parameter.

property full_label
The label of the parameter with its path in a parameter group prepended.

get_value_and_bounds_for_optimization() → tuple[float, float, float]
Gets the parameter value and bounds with expression and non-negative constraints applied.

property label
Label of the parameter

property maximum
The upper bound of the parameter.

property minimum
The lower bound of the parameter.

property non_negative
Indicates if the parameter is non-negativ.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Always *False* if *expression* is not *None*.

set_from_group(*group: ParameterGroup*)
Sets all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** – The `glotaran.parameter.ParameterGroup`.

set_value_from_optimization(*value: float*)
Sets the value from an optimization result and reverses non-negative transformation.

property standard_error
The standard error of the optimized parameter.

property transformed_expression
The expression of the parameter transformed for evaluation within a *ParameterGroup*.

classmethod valid_label(*label: str*) → bool
Returns true if the *label* is valid string.

property value
The value of the parameter

property vary
Indicates if the parameter should be optimized.

Always *False* if *expression* is not *None*.

parameter_group

The parameter group class

Classes

Summary

ParameterGroup	Represents are group of parameters.

ParameterGroup

class glotaran.parameter.parameter_group.**ParameterGroup**(*() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*)

Bases: dict

Represents are group of parameters. Can contain other groups, creating a tree-like hierarchy.

Parameters **label** – The label of the group.

Attributes Summary

label	Label of the group.
root_group	Root of the group.

label

ParameterGroup.**label**
Label of the group.

root_group

ParameterGroup.**root_group**
Root of the group.

Methods Summary

add_group	Adds a *ParameterGroup* to the group.
add_parameter	Adds a Parameter to the group.
all	Returns a generator over all parameter in the group and it's subgroups together with their labels.
clear	
copy	
from_csv	Creates a *ParameterGroup* from a CSV file.
from_dataframe	Creates a *ParameterGroup* from a pandas.DataFrame
from_dict	Creates a *ParameterGroup* from a dictionary.
from_file	
from_list	Creates a *ParameterGroup* from a list.
from_yaml	Creates a *ParameterGroup* from a YAML string.
from_yaml_file	Creates a *ParameterGroup* from a YAML file.
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Gets a Parameter by its label.
get_label_value_and_bounds_arrays	Returns a arrays of all parameter labels, values and bounds.
get_nr_roots	Returns the number of roots of the group.
groups	Returns a generator over all groups and their subgroups.
has	Checks if a parameter with the given label is in the group or in a subgroup.
items	
keys	
known_formats	

continues on next page

Table 147 – continued from previous page

markdown	Formats the *ParameterGroup* as markdown string.
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
set_from_label_and_value_arrays	Updates the parameter values from a list of labels and values.
setdefault	Insert key with a value of default if key is not in the dictionary.
to_csv	Writes a *ParameterGroup* to a CSV file.
to_dataframe	
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
update_parameter_expression	Updates all parameters which have an expression.
values	

add_group

ParameterGroup.**add_group**(*group:* glotaran.parameter.parameter_group.ParameterGroup)

Adds a *ParameterGroup* to the group.

Parameters **group** – The group to add.

add_parameter

ParameterGroup.**add_parameter**(*parameter: Parameter | list[Parameter]*)

Adds a Parameter to the group.

Parameters **parameter** – The parameter to add.

all

ParameterGroup.**all**(*root: str = None*, *separator: str = '.'*) → Generator[tuple[str, Parameter], None, None]

Returns a generator over all parameter in the group and it's subgroups together with their labels.

Parameters

- **root** – The label of the root group
- **separator** – The separator for the parameter labels.

clear

ParameterGroup.**clear**() → None. Remove all items from D.

copy

ParameterGroup.**copy**() → a shallow copy of D

from_csv

classmethod ParameterGroup.**from_csv**(*filepath: str*, *delimiter: Optional[str] = None*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a *ParameterGroup* from a CSV file.

Parameters

- **filepath** – The path to the CSV file.
- **delimiter** – The delimiter of the CSV file.

from_dataframe

classmethod ParameterGroup.**from_dataframe**(*df: pandas.core.frame.DataFrame*, *source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a *ParameterGroup* from a `pandas.DataFrame`

from_dict

classmethod ParameterGroup.**from_dict**(*parameter_dict: dict[str, dict | list]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Creates a *ParameterGroup* from a dictionary.

Parameters

- **parameter_dict** – A parameter dictionary containing parameters.
- **label** – The label of root group.
- **root_group** – The root group

from_file

classmethod ParameterGroup.**from_file**(*filepath: str*, *fmt: Optional[str] = None*)

from_list

classmethod ParameterGroup.**from_list**(*parameter_list: list[float | list]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Creates a *ParameterGroup* from a list.

Parameters

- **parameter_list** – A parameter list containing parameters
- **label** – The label of the root group.
- **root_group** – The root group

from_yaml

classmethod ParameterGroup.**from_yaml**(*yaml_string: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a *ParameterGroup* from a YAML string.

Parameters **yaml_string** – The YAML string with the parameters.

from_yaml_file

classmethod ParameterGroup.**from_yaml_file**(*filepath: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a *ParameterGroup* from a YAML file.

Parameters **filepath** – The path to the YAML file.

fromkeys

ParameterGroup.**fromkeys**(*iterable*, *value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get

ParameterGroup.**get**(*label: str*) → *glotaran.parameter.parameter.Parameter*

Gets a `Parameter` by its label.

Parameters **label** – The label of the parameter.

get_label_value_and_bounds_arrays

ParameterGroup.**get_label_value_and_bounds_arrays**(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

Returns a arrays of all parameter labels, values and bounds.

Parameters **exclude_non_vary** (*bool = False*) – If true, parameters with *vary=False* are excluded.

get_nr_roots

ParameterGroup.**get_nr_roots**() → int
Returns the number of roots of the group.

groups

ParameterGroup.**groups**() → Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]
Returns a generator over all groups and their subgroups.

has

ParameterGroup.**has**(*label: str*) → bool
Checks if a parameter with the given label is in the group or in a subgroup.
Parameters **label** – The label of the parameter.

items

ParameterGroup.**items**() → a set-like object providing a view on D's items

keys

ParameterGroup.**keys**() → a set-like object providing a view on D's keys

known_formats

classmethod ParameterGroup.**known_formats**() → dict[str, Callable]

markdown

ParameterGroup.**markdown**() → str
Formats the *ParameterGroup* as markdown string.

pop

ParameterGroup.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem

ParameterGroup.**popitem**(/)
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

set_from_label_and_value_arrays

ParameterGroup.**set_from_label_and_value_arrays**(*labels: list[str]*, *values: np.ndarray*)
Updates the parameter values from a list of labels and values.

setdefault

ParameterGroup.**setdefault**(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv

ParameterGroup.**to_csv**(*filename: str*, *delimiter: str = ','*)
Writes a *ParameterGroup* to a CSV file.
Parameters
- **filepath** – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe

ParameterGroup.**to_dataframe**() → pandas.core.frame.DataFrame

update

ParameterGroup.**update**([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression

ParameterGroup.**update_parameter_expression**()
Updates all parameters which have an expression.

values

ParameterGroup.**values**() → an object providing a view on D's values

Methods Documentation

add_group(*group:* glotaran.parameter.parameter_group.ParameterGroup)
Adds a *ParameterGroup* to the group.
Parameters **group** – The group to add.

add_parameter(*parameter: Parameter | list[Parameter]*)
Adds a Parameter to the group.
Parameters **parameter** – The parameter to add.

all(*root: str = None*, *separator: str = '.'*) → Generator[tuple[str, Parameter], None, None]
Returns a generator over all parameter in the group and it's subgroups together with their labels.
Parameters
- **root** – The label of the root group
- **separator** – The separator for the parameter labels.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

classmethod from_csv(*filepath: str*, *delimiter: Optional[str] = None*) → *glotaran.parameter.parameter_group.ParameterGroup*
Creates a *ParameterGroup* from a CSV file.
Parameters
- **filepath** – The path to the CSV file.
- **delimiter** – The delimiter of the CSV file.

classmethod from_dataframe(*df: pandas.core.frame.DataFrame*, *source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*
Creates a *ParameterGroup* from a pandas.DataFrame

classmethod from_dict(*parameter_dict: dict[str, dict | list]*, *label: str = None*, *root_group:* ParameterGroup *= None*) → *ParameterGroup*
Creates a *ParameterGroup* from a dictionary.
Parameters
- **parameter_dict** – A parameter dictionary containing parameters.
- **label** – The label of root group.
- **root_group** – The root group

classmethod from_file(*filepath: str*, *fmt: Optional[str] = None*)

classmethod from_list(*parameter_list: list[float | list]*, *label: str = None*, *root_group:* ParameterGroup *= None*) → *ParameterGroup*
Creates a *ParameterGroup* from a list.
Parameters
- **parameter_list** – A parameter list containing parameters
- **label** – The label of the root group.
- **root_group** – The root group

classmethod from_yaml(*yaml_string: str*) → *glotaran.parameter.parameter_group.ParameterGroup*
Creates a *ParameterGroup* from a YAML string.
Parameters **yaml_string** – The YAML string with the parameters.

classmethod from_yaml_file(*filepath: str*) → *glotaran.parameter.parameter_group.ParameterGroup*
Creates a `ParameterGroup` from a YAML file.
Parameters **filepath** – The path to the YAML file.

fromkeys(*iterable*, *value=None*, */*)
Create a new dictionary with keys from iterable and values set to value.

get(*label: str*) → *glotaran.parameter.parameter.Parameter*
Gets a `Parameter` by its label.
Parameters **label** – The label of the parameter.

get_label_value_and_bounds_arrays(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]
Returns a arrays of all parameter labels, values and bounds.
Parameters **exclude_non_vary** (`bool = False`) – If true, parameters with *vary=False* are excluded.

get_nr_roots() → int
Returns the number of roots of the group.

groups() → Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]
Returns a generator over all groups and their subgroups.

has(*label: str*) → bool
Checks if a parameter with the given label is in the group or in a subgroup.
Parameters **label** – The label of the parameter.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

classmethod known_formats() → dict[str, Callable]

property label
Label of the group.

markdown() → str
Formats the `ParameterGroup` as markdown string.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem(*/*)
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

property root_group
Root of the group.

set_from_label_and_value_arrays(*labels: list[str]*, *values: np.ndarray*)
Updates the parameter values from a list of labels and values.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv(*filename: str*, *delimiter: str = ','*)
Writes a `ParameterGroup` to a CSV file.
Parameters
- **filepath** – The path to the CSV file.

- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe() → pandas.core.frame.DataFrame

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression()
Updates all parameters which have an expression.

values() → an object providing a view on D's values

Exceptions

Exception Summary

ParameterNotFoundException	Raised when a Parameter is not found in the Group.

ParameterNotFoundException

exception glotaran.parameter.parameter_group.**ParameterNotFoundException**(*path*, *label*)

Raised when a Parameter is not found in the Group.

12.1.8 parse

Glotarans parsing package

Modules

glotaran.parse.parser	Functions for reading and parsing models from serialized representations.
glotaran.parse.register	A register for models
glotaran.parse.util	

parser

Functions for reading and parsing models from serialized representations.

Functions

Summary

load_yaml	
load_yaml_file	
parse_spec	
parse_yaml	
parse_yaml_file	parse_yaml_file reads the given file and parses its content as YML.

load_yaml

glotaran.parse.parser.**load_yaml**(*data: str*)

load_yaml_file

glotaran.parse.parser.**load_yaml_file**(*filename: str*)

parse_spec

glotaran.parse.parser.**parse_spec**(*spec: Dict*)

parse_yaml

glotaran.parse.parser.**parse_yaml**(*data: str*)

parse_yaml_file

glotaran.parse.parser.**parse_yaml_file**(*filename: str*) → Dict

parse_yaml_file reads the given file and parses its content as YML.

Parameters **filename** (*str*) – filename is the of the file to parse.

Returns **content** – The content of the file as dictionary.

Return type Dict

register

A register for models

Functions

Summary

get_model	get_model gets a model from the register.
known_model	known_model returns True if the model_type is in the register.
known_model_names	
register_model	register_model registers a model.

get_model

glotaran.parse.register.**get_model**(*model_type: str*) → Model
get_model gets a model from the register.

Parameters **model_type** – model_type is type of the model.

known_model

glotaran.parse.register.**known_model**(*model_type: str*) → bool
known_model returns True if the model_type is in the register.

Parameters **model_type** – model_type is type of the model.

known_model_names

glotaran.parse.register.**known_model_names**() → list[str]

register_model

glotaran.parse.register.**register_model**(*model_type: str*, *model: Model*)
register_model registers a model.

Parameters

- **model_type** – model_type is type of the model.
- **model** – model is the model to be registered.

util

Functions

Summary

list_string_to_tuple	Converts a list of strings (representing tuples) to a list of tuples
sanitize_dict_keys	Sanitize the stringified tuple dict keys in a yaml parsed dict
sanitize_dict_values	Sanitizes a dict with broken tuples inside modifying it in-place Broken tuples are tuples that are turned into strings by the yaml parser.
sanitize_list_with_broken_tuples	Sanitize a list with 'broken' tuples
sanitize_yaml	Sanitize a yaml-returned dict for key or (list) values containing tuples
string_to_tuple	[summary]

list_string_to_tuple

glotaran.parse.util.**list_string_to_tuple**(*a_list: List[str]*) → List[Union[str, float]]

Converts a list of strings (representing tuples) to a list of tuples

Parameters **a_list** (*List[str]*) – A list of strings, some of them representing (numbered) tuples

Returns A list of the (numbered) tuples represted by the incoming a_list

Return type List[Union[float, str]]

sanitize_dict_keys

glotaran.parse.util.**sanitize_dict_keys**(*d: dict*) → dict

Sanitize the stringified tuple dict keys in a yaml parsed dict

Keys representing a tuple, e.g. '(s1, s2)' are converted to a tuple of strings e.g. ('s1', 's2')

Parameters **d** (*dict*) – A dict containing tuple-like string keys

Returns A dict with tuple-like string keys converted to tuple keys

Return type dict

sanitize_dict_values

glotaran.parse.util.**sanitize_dict_values**(*d: dict*)

Sanitizes a dict with broken tuples inside modifying it in-place Broken tuples are tuples that are turned into strings by the yaml parser. This functions calls *sanitize_list_with_broken_tuples* to glue the broken strings together and then calls list_to_tuple to turn the list with tuple strings back to number tuples.

Args: d (dict): A (complex) dict containing (possibly nested) values of broken tuple strings

sanitize_list_with_broken_tuples

glotaran.parse.util.**sanitize_list_with_broken_tuples**(*mangled_list: List[Union[str, float]]*) → List[str]

Sanitize a list with 'broken' tuples

A list of broken tuples as returned by yaml when parsing tuples. e.g parsing the list of tuples [(3,100), (4,200)] results in a list of str ['(3', '100)', '(4', '200)'] which can be restored to a list with the tuples restored as strings ['(3, 100)', '(4, 200)']

Parameters **mangled_list** (*List[Union[str, float]]*) – A list with strings representing tuples broken up by round brackets.

Returns A list containing the restores tuples (in string form) which can be converted back to numbered tuples using *list_string_to_tuple*

Return type List[str]

sanitize_yaml

glotaran.parse.util.**sanitize_yaml**(*d: dict*, *do_keys: bool = True*, *do_values: bool = False*) → dict

Sanitize a yaml-returned dict for key or (list) values containing tuples

Parameters **d** (*dict*) – a dict resulting from parsing a pyglotaran model spec yml file

Returns a sanitized dict with (broken) string tuples restored as proper tuples

Return type dict

string_to_tuple

glotaran.parse.util.**string_to_tuple**(*tuple_str: str*, *from_list=False*) → Union[Tuple[float], Tuple[str], float, str]

[summary]

Parameters

- **tuple_str** (*str*) – A string representing some tuple to convert the numbers inside the string tuple are mapped to float
- **from_list** (*bool, optional*) – only if true will a single number string be converted to float, otherwise returned as-is since it may represent a label, by default False

Returns Returns the tuple intended by the string

Return type Union[Tuple[float], Tuple[str], float, str]

CHAPTER

THIRTEEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

13.1 Types of Contributions

13.1.1 Report Bugs

Report bugs at https://github.com/glotaran/pyglotaran/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

13.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

13.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

13.1.4 Write Documentation

pyglotaran could always use more documentation, whether as part of the official pyglotaran docs, in docstrings, or even on the web in blog posts, articles, and such. If you are writing docstrings please use the NumPyDoc style to write them.

13.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/glotaran/pyglotaran/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

13.2 Get Started!

Ready to contribute? Here's how to set up `pyglotaran` for local development.

1. Fork the `pyglotaran` repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/<your_name_here>/pyglotaran.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyglotaran
(pyglotaran)$ cd pyglotaran
(pyglotaran)$ python -m pip install -r requirements_dev.txt
(pyglotaran)$ pip install -e . --process-dependency-links
```

4. Install the `pre-commit` hooks, to automatically format and check your code:

```
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ pre-commit run -a
$ py.test
```

   Or to run all at once:

```
$ tox
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

13.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a *docstring*.
3. The pull request should work for Python 3.8 Check your Github Actions `https://github.com/<your_name_here>/pyglotaran/actions` and make sure that the tests pass for all supported Python versions.

13.4 Docstrings

We use numpy style docstrings, which can also be autogenerated from function/method signatures by extensions for your editor.

Some extensions for popular editors are:

- autodocstring (VS-Code)
- vim-python-docstring (Vim)

Note: If your pull request improves the docstring coverage (check `pre-commit run -a interrogate`), please raise the value of the interrogate setting `fail-under` in pyproject.toml. That way the next person will improve the docstring coverage as well and everyone can enjoy a better documentation.

Warning: As soon as all our docstrings in proper shape we will enforce that it stays that way. If you want to check if your docstrings are fine you can use pydocstyle and darglint.

13.5 Tips

To run a subset of tests:

```
$py.test tests.test_pyglotaran
```

13.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Github Actions will then deploy to PyPI if the tests pass.

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] https://glotaran.github.io/legacy/file_formats

PYTHON MODULE INDEX

g

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z