
pyglotaran Documentation

Release 0.3.3

Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum

Mar 18, 2021

CONTENTS:

1	Introduction	1
2	Installation	3
3	Quickstart/Cheat-Sheet	5
4	History	13
5	Authors	15
6	Overview	17
7	Data IO	19
8	Plotting	21
9	Modelling	23
10	Parameter	25
11	Optimizing	27
12	API Documentation	29
13	Contributing	221
14	Indices and tables	225
	Bibliography	227
	Python Module Index	229
	Index	231

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 (`*.yaml`) 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.

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Warning: pyglotaran is early development, so for the moment stable releases are sparse and outdated.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 re-run the command above.

QUICKSTART/CHEAT-SHEET

Warning: This is documentation for an early access release of pyglotaran, certain aspects of how it works (as demonstrated in this quickstart) may be subjected to changes in future 0.x releases of the software. Please consult the [pyglotaran readme](#) to learn more on what this means.

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 0.01551 0.02087 0.001702 ... 1.714 1.538
```

Like all data in pyglotaran, the dataset is a `xarray.Dataset`. You can find more information about the `xarray` library the [xarray homepage](#).

The loaded dataset is a simulated sequential model.

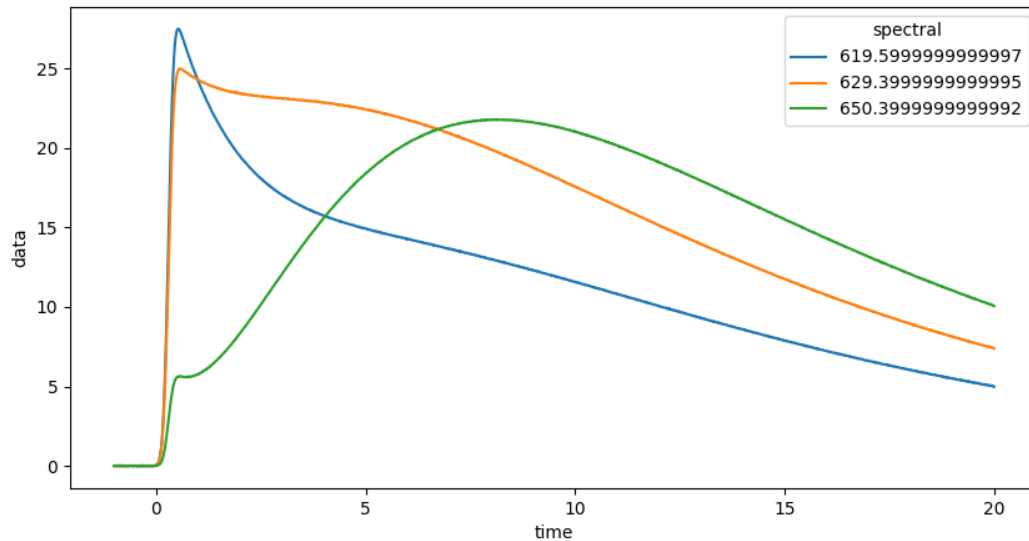
To plot our data, we must first import matplotlib.

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

Now we can plot some time traces.

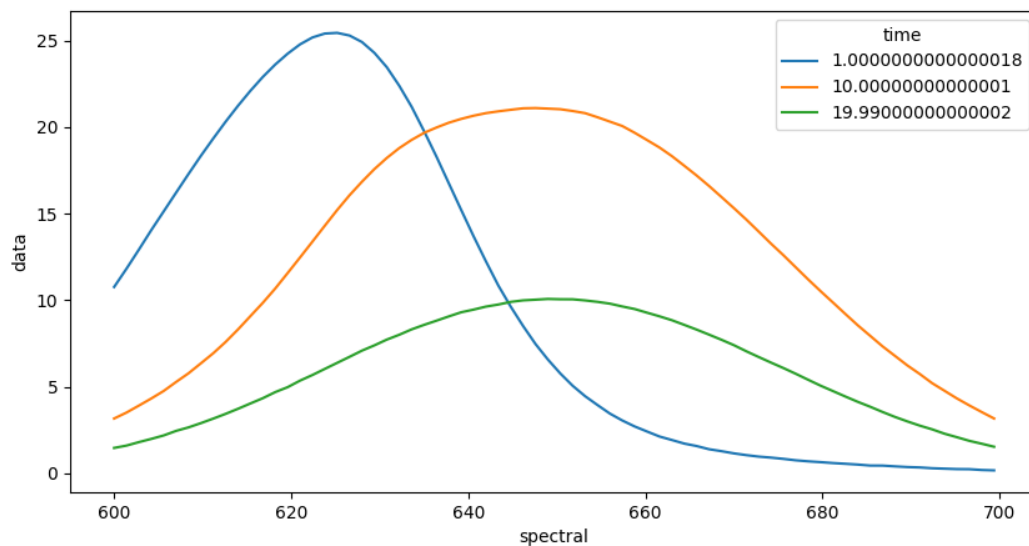
```
In [7]: plot_data = dataset.data.sel(spectral=[620, 630, 650], method='nearest')

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: 72, 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 0.01551 ... 1.538
  data_left_singular_vectors            (time, left_singular_value_index) float64 -8...
  data_singular_values                  (singular_value_index) float64 4.62e+03 ... ...
  data_right_singular_vectors           (right_singular_value_index, spectral) float64 ...

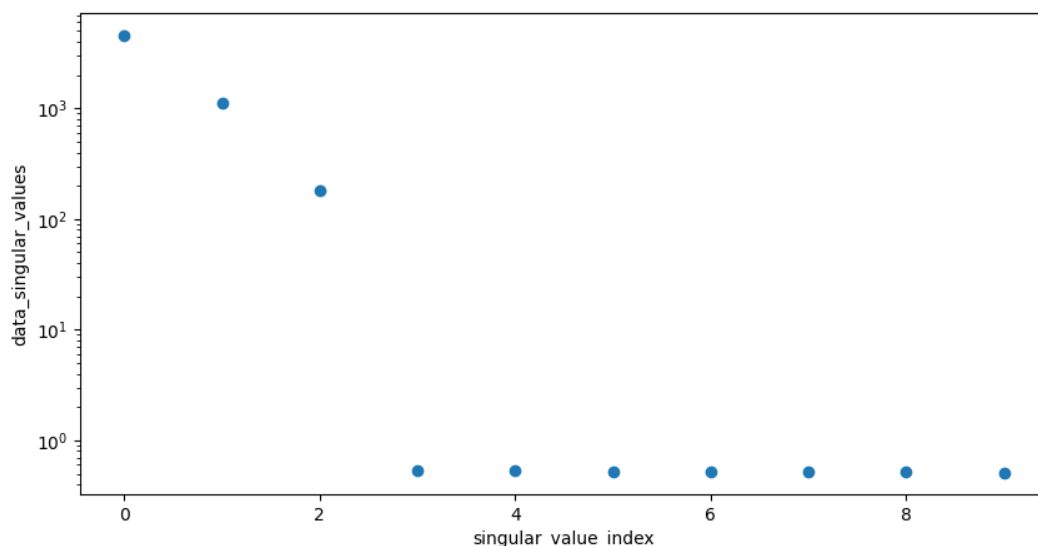
```

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.yml` 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      7.5381e+00              3.
↳74e+01
      1              2      7.5377e+00      4.62e-04      1.02e-04      3.28e-
↳01
      2              3      7.5377e+00      9.13e-10      3.48e-08      3.49e-
↳06
`ftol` termination condition is satisfied.
Function evaluations 3, initial cost 7.5381e+00, final cost 7.5377e+00, first-order_
↳optimality 3.49e-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 |    1.51e+01 |
Reduced Chi Square |    9.97e-05 |
Root Mean Square Error (RMSE) |    9.99e-03 |

```

```
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_: 0.4999391441068187, _StdErr_: 0.007261698651628417, _Min_: -inf,
↳_Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
* __2__: _Value_: 0.30008042937966384, _StdErr_: 0.0041961923515427, _Min_: -inf, _
↳_Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
* __3__: _Value_: 0.0999874440333988, _StdErr_: 0.0004778304467913974, _Min_: -
↳inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
* __irf__:
* __center__: _Value_: 0.30000199460726046, _StdErr_: 0.0005010729238540931, _Min_
↳: -inf, _Max_: inf, _Vary_: True, _Non-Negative_: False, _Expr_: None
* __width__: _Value_: 0.09999158675249081, _StdErr_: 0.0006703684374783712, _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`.

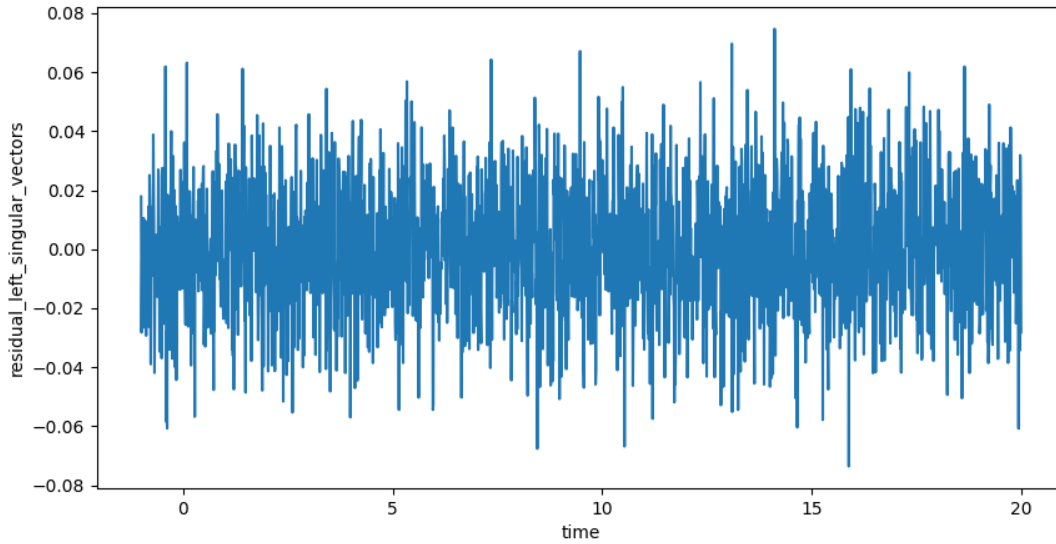
```
In [25]: result_dataset = result.get_dataset('dataset1')

In [26]: result_dataset
Out[26]:
<xarray.Dataset>
Dimensions:                                (clp_label: 3, component: 3, from_
↳species: 3, left_singular_value_index: 72, 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: (12/24)
  data                                (time, spectral) float64 0.0155...
  data_left_singular_vectors          (time, left_singular_value_index)↳
↳float64 ...
  data_singular_values                (singular_value_index) float64 ...
  data_right_singular_vectors         (right_singular_value_index, spectral)↳
↳float64 ...
  matrix                              (time, clp_label) float64 6.006...
  clp                                 (spectral, clp_label) float64 1...
  ...
  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 0.09999
  irf                                 (time) float64 1.976e-37 ... 0.0
Attributes:
  root_mean_square_error:              0.009985218742264433
  weighted_root_mean_square_error:    0.009985218742264433
```

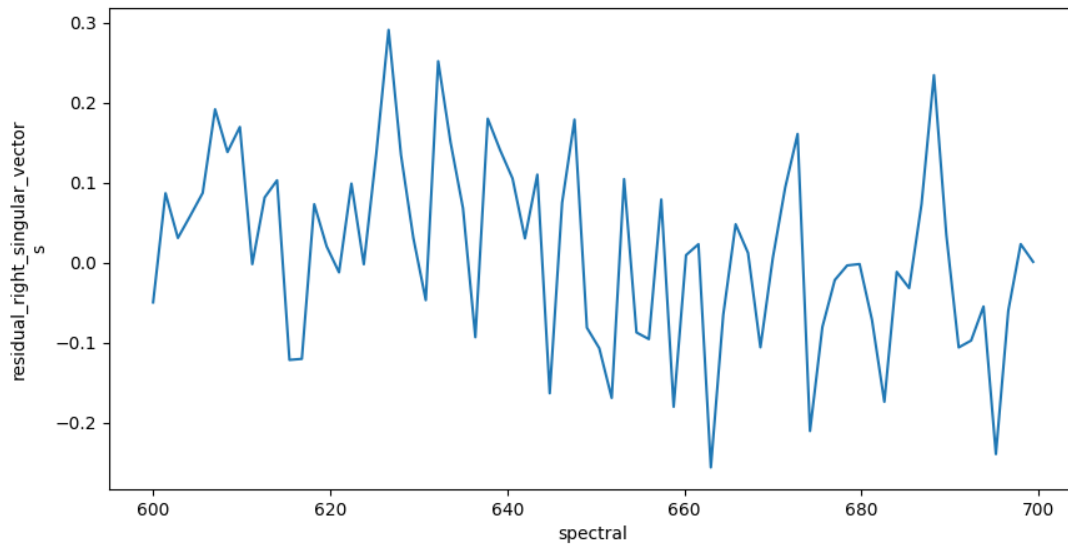
The resulting data can be visualized the same way as the dataset. To judge the quality of the fit, you should look at first left and right singular vectors of the residual.

```
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')
```


HISTORY

4.1 0.3.3 (2021-03-18)

- Force recalculation of SVD attributes in `scheme._prepare_data` (#597)
- Remove unneeded check in `spectral_penalties._get_area` Fixes (#598)
- Added python 3.9 support (#450)

4.2 0.3.2 (2021-02-28)

- Re-release of version 0.3.1 due to packaging issue

4.3 0.3.1 (2021-02-28)

- Added compatibility for numpy 1.20 and raised minimum required numpy version to 1.20 (#555)
- Fixed excessive memory consumption in result creation due to full SVD computation (#574)
- Added feature parameter history (#557)
- Moved setup logic to `setup.cfg` (#560)

4.4 0.3.0 (2021-02-11)

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

4.5 0.2.0 (2020-12-02)

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

4.6 0.1.0 (2020-07-14)

- Package was renamed to pyglotaran on PyPi

4.7 0.0.8 (2018-08-07)

- Changed `nan_policy` to `omit`

4.8 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartement.

4.9 0.0.6 (2018-08-07)

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

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. Laptanok

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, Laptanok 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.bbabi.2004.04.011>

OVERVIEW

CHAPTER
SEVEN

DATA IO

PLOTTING

MODELLING

PARAMETER

OPTIMIZING

API DOCUMENTATION

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

<i>glotaran</i>	Glotaran package <code>__init__.py</code>
-----------------	---

12.1 glotaran

Glotaran package `__init__.py`

Modules

<i>glotaran.analysis</i>	This package contains functions for model simulation and fitting.
--------------------------	---

<i>glotaran.builtin</i>	
-------------------------	--

<i>glotaran.cli</i>	
---------------------	--

<i>glotaran.examples</i>	
--------------------------	--

<i>glotaran.io</i>	Functions for data IO
<i>glotaran.model</i>	Glotaran Model Package
<i>glotaran.parameter</i>	

<i>glotaran.parse</i>	Glotarans parsing package
-----------------------	---------------------------

12.1.1 analysis

This package contains functions for model simulation and fitting.

Modules

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

<code>residual_nnls</code>	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, descrip-  
                                              tor)
```

Bases: `tuple`

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

Attributes Summary

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

<code>count</code>	Return number of occurrences of value.
<code>index</code>	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

<i>axis</i>	Alias for field number 2
<i>index</i>	Alias for field number 1
<i>label</i>	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

<i>count</i>	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

<code>clp_label</code>	Alias for field number 0
<code>matrix</code>	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

<code>count</code>	Return number of occurrences of value.
<code>index</code>	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

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

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`

parameter_history

Problem.**parameter_history**

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

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

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, history_index: Optional[int] =
                           None) → Dict[str, xarray.core.dataset.Dataset]
```

reset

```
Problem.reset()
```

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

save_parameters_for_history

```
Problem.save_parameters_for_history()
```

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() → Tu-
ple[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, history_index: Optional[int] = None) →
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 `parameter_history`

property `parameters`

property `reduced_clp_labels`

property `reduced_clps`

property `reduced_matrices`

reset ()

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

property `residuals`

save_parameters_for_history ()

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

<code>data</code>	Alias for field number 1
<code>dataset</code>	Alias for field number 0
<code>global_axis</code>	Alias for field number 3
<code>model_axis</code>	Alias for field number 2
<code>weight</code>	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

<code>count</code>	Return number of occurrences of value.
<code>index</code>	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

<code>additional_penalty</code>	The additional penalty vector.
<code>chi_square</code>	The chi-square of the optimization.
<code>covariance_matrix</code>	Covariance matrix.
<code>data</code>	The resulting data as a dictionary of <code>xarray.Dataset</code> .
<code>degrees_of_freedom</code>	Degrees of freedom in optimization $N - N_{vars}$.
<code>free_parameter_labels</code>	List of labels of the free parameters used in optimization.
<code>initial_parameters</code>	The initial parameters.
<code>jacobian</code>	Modified Jacobian matrix at the solution See also: <code>scipy.optimize.least_squares()</code>
<code>model</code>	The model for analysis.
<code>nnls</code>	If <code>True</code> non-negative least squares optimization is used instead of the default variable projection.
<code>number_of_data_points</code>	Number of data points N .
<code>number_of_function_evaluations</code>	The number of function evaluations.

continues on next page

Table 19 – continued from previous page

<i>number_of_jacobian_evaluations</i>	The number of jacobian evaluations.
<i>number_of_variables</i>	Number of variables in optimization N_{vars}
<i>optimized_parameters</i>	The optimized parameters.
<i>reduced_chi_square</i>	The reduced chi-square of the optimization.
<i>root_mean_square_error</i>	The root mean square error the optimization.
<i>scheme</i>	The scheme for analysis.
<i>success</i>	Indicates if the optimization was successful.
<i>termination_reason</i>	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 initial 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.

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

root_mean_square_error

`Result.root_mean_square_error`

The root mean square error the optimization.

$$rms = \sqrt{\chi_{red}^2}$$

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

<code>get_dataset</code>	Returns the result dataset for the given dataset label.
<code>get_scheme</code>	Return a new scheme from the Result object with optimized parameters.
<code>markdown</code>	Formats the model as a markdown text.
<code>save</code>	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.

$$\chi^2 = \sum_i^N [\text{Residual}_i]^2.$$

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 initial 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.

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

property root_mean_square_error

The root mean square error the optimization.

$$rms = \sqrt{\chi_{red}^2}$$

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`

`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

<code>from_yaml_file</code>	
<code>markdown</code>	
<code>problem_list</code>	Returns a list with all problems in the model and missing parameters.
<code>valid</code>	Returns <i>True</i> if there are no problems with the model or the parameters, else <i>False</i> .
<code>validate</code>	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

<code>simulate</code>	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

<code>residual_variable_projection</code>	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`

<code>read_ascii_time_trace</code>	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

<i>DataFileType</i>	An enumeration.
<i>ExplicitFile</i>	Abstract class representing either a time- or wavelength-explicit file.
<i>TimeExplicitFile</i>	Represents a time explicit file

continues on next page

Table 30 – continued from previous page

<i>WavelengthExplicitFile</i>	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

<i>time_explicit</i>
<i>wavelength_explicit</i>

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
da
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 ex-
      plicit'>, 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`*) \rightarrow `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(axes)
```

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(axes)
```

```
write(overwrite=False, comment="", file_format=<DataFileType.time_explicit: 'Time ex-  
plicit'>, number_format='%.10e')
```

WavelengthExplicitFile

class `glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile`

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`

`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

Glutarans module to read files

Functions

Summary

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

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

<code>glotaran.builtin.models.kinetic_image.initial_concentration</code>	This package contains the initial concentration item.
--	---

<code>glotaran.builtin.models.kinetic_image.irf</code>	This package contains irf items.
--	----------------------------------

<code>glotaran.builtin.models.kinetic_image.k_matrix</code>	K-Matrix
---	----------

<code>glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor</code>	Kinetic Image Dataset Descriptor
---	----------------------------------

<code>glotaran.builtin.models.kinetic_image.kinetic_image_matrix</code>	Glotaran Kinetic Matrix
---	-------------------------

<code>glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex</code>	This package contains the kinetic megacomplex item.
--	---

<code>glotaran.builtin.models.kinetic_image.kinetic_image_model</code>	
--	--

<code>glotaran.builtin.models.kinetic_image.kinetic_image_result</code>	
---	--

initial_concentration

This package contains the initial concentration item.

Classes

Summary

<i>InitialConcentration</i>	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

<i>compartments</i>
<i>exclude_from_normalize</i>
<i>label</i>
<i>parameters</i>

compartments

`InitialConcentration.compartments`

exclude_from_normalize

`InitialConcentration.exclude_from_normalize`

label

`InitialConcentration.label`

parameters

`InitialConcentration.parameters`

Methods Summary

<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>mprint</i>	
<i>normalized</i>	
<i>validate</i>	

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.InitialConcentration*

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

<i>Irf</i>	Represents an IRF.
<i>IrfGaussian</i>	
<i>IrfMeasured</i>	A measured IRF.
<i>IrfMultiGaussian</i>	Represents a gaussian IRF.

Irf

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

Bases: `object`

Represents an IRF.

Methods Summary

<i>add_type</i>

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

<i>backsweep</i>
<i>backsweep_period</i>
<i>center</i>
<i>label</i>
<i>normalize</i>
<i>scale</i>
<i>type</i>
<i>width</i>

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**

<code>calculate</code>	
<code>fill</code>	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.
<code>from_dict</code>	
<code>from_list</code>	
<code>mprint</code>	
<code>parameter</code>	
<code>validate</code>	

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

<i>label</i>
<i>type</i>

label

IrfMeasured.**label**

type

IrfMeasured.**type**

Methods Summary

<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>mprint</i>	
<i>validate</i>	

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 `backswap`

property `backswap_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 differential system.

KMatrix

class `glotaran.builtin.models.kinetic_image.k_matrix.KMatrix`

Bases: `object`

A K-Matrix represents a first order differential system.

Attributes Summary

label

matrix

label

`KMatrix.label`

matrix

`KMatrix.matrix`

Methods Summary

<i>a_matrix</i>	The resulting A matrix of the KMatrix.
<i>a_matrix_as_markdown</i>	Returns the A Matrix as markdown formatted table.
<i>a_matrix_non_unibranched</i>	The resulting A matrix of the KMatrix for a non-unibranched model.
<i>a_matrix_unibranched</i>	The resulting A matrix of the KMatrix for an unibranched model.
<i>combine</i>	Creates a combined matrix.
<i>eigen</i>	Returns the eigenvalues and eigenvectors of the k matrix.
<i>empty</i>	Creates an empty K-Matrix.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>full</i>	The full representation of the KMatrix as numpy array.
<i>involved_compartments</i>	A list of all compartments in the Matrix.
<i>is_unibranched</i>	Returns true in the KMatrix represents an unibranched model.

continues on next page

Table 52 – continued from previous page

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

a_matrix

KMatrix.**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

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_unibranh

KMatrix.**a_matrix_non_unibranh** (*initial_concentration*: [glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration](#)
→ [numpy.ndarray](#))

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

Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranh

KMatrix.**a_matrix_unibranh** (*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_unbranched

`KMatrix.is_unbranched (initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration) → bool`

Returns true if the KMatrix represents an unbranched 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.InitialConcentration) → str`

Returns the A Matrix as markdown formatted table.

Parameters `initial_concentration` – The initial concentration.

`a_matrix_non_unibranched(initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration) → numpy.ndarray`

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

Parameters `initial_concentration` – The initial concentration.

`a_matrix_unibranched(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(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_unbranched (*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → bool

Returns true in the KMatrix represents an unbranched 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.KineticImageDatasetDescriptor`
 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

<i>compartments</i>	
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>get_k_matrices</i>	
<i>mprint</i>	
<i>validate</i>	

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

<code>calculate_kinetic_matrix_gaussian_irf</code>	Calculates a kinetic matrix with a gaussian irf.
<code>calculate_kinetic_matrix_no_irf</code>	
<code>kinetic_image_matrix_implementation</code>	
<code>kinetic_matrix</code>	

`calculate_kinetic_matrix_gaussian_irf`

`glotaran.builtin.models.kinetic_image.kinetic_image_matrix.calculate_kinetic_matrix_gaussian_irf`

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_irf`

`kinetic_image_matrix_implementation`

`glotaran.builtin.models.kinetic_image.kinetic_image_matrix.kinetic_image_matrix_implementation`

kinetic_matrix

glotaran.builtin.models.kinetic_image.kinetic_image_matrix.**kinetic_matrix** (*dataset_descriptor*,
axis=None,
in-
dex=None,
irf=None,
ma-
trix_implementation)

kinetic_image_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

<i>KineticImageMegacomplex</i>	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

<i>involved_compartments</i>
<i>k_matrix</i>
<i>label</i>
<i>scale</i>

involved_compartments

KineticImageMegacomplex.**involved_compartments**

k_matrix

KineticImageMegacomplex.**k_matrix**

label

KineticImageMegacomplex.**label**

scale

KineticImageMegacomplex.**scale**

Methods Summary

<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>full_k_matrix</i>	
<i>mprint</i>	
<i>validate</i>	

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_definition*, *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

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

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

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

get_initial_concentration

`KineticImageModel.get_initial_concentration` (*label*: `str`) → `glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration`

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.KineticImageMegacomplex`

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.KineticImageDatasetDescriptor)`

set_initial_concentration

`KineticImageModel.set_initial_concentration(label: str, item: glotaran.builtin.models.kinetic_image.initial_concentration_descriptor.InitialConcentrationDescriptor)`

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.KineticImageMegaComplex)

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.KineticImageDatasetDescriptor)
set_initial_concentration (label: str, item: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentrationDescriptor)
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.KineticImageMegacomplexDescriptor)
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_associated_data
```

```
retrieve_irf
```

```
retrieve_species_associated_data
```

finalize_kinetic_image_result

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

retrieve_decay_associiated_data

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_decay_associiated**

retrieve_irf

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

retrieve_species_associiated_data

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_species_associiated**

kinetic_spectrum

Modules

<i>glotaran.builtin.models. kinetic_spectrum. kinetic_spectrum_dataset_descriptor</i>	
<i>glotaran.builtin.models. kinetic_spectrum. kinetic_spectrum_matrix</i>	Glotaran kinetic spectrum Matrix
<i>glotaran.builtin.models. kinetic_spectrum. kinetic_spectrum_model</i>	
<i>glotaran.builtin.models. kinetic_spectrum. kinetic_spectrum_result</i>	
continues on next page	

Table 65 – continued from previous page

<i>glotaran.builtin.models. kinetic_spectrum.spectral_constraints</i>	This package contains compartment constraint items.
<i>glotaran.builtin.models. kinetic_spectrum.spectral_irf</i>	
<i>glotaran.builtin.models. kinetic_spectrum.spectral_matrix</i>	Glotaran Spectral Matrix
<i>glotaran.builtin.models. kinetic_spectrum.spectral_penalties</i>	This package contains compartment constraint items.
<i>glotaran.builtin.models. kinetic_spectrum.spectral_relations</i>	Glotaran Spectral Relation
<i>glotaran.builtin.models. kinetic_spectrum.spectral_shape</i>	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.KineticSpectrumDatasetDescriptor`
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

<i>grouped</i>	
<i>has_kinetic_model_constraints</i>	
<i>has_spectral_penalties</i>	
<i>index_dependent</i>	
<i>kinetic_spectrum_matrix</i>	
<i>retrieve_spectral_clps</i>	
<i>spectral_matrix</i>	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_penalties`

grouped

`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.grouped` (*model:*
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model)

has_kinetic_model_constraints

`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.has_kinetic_model_constraints`

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.builtin.models.kinetic_spectrum.kinetic_spectrum_model
 \rightarrow
bool)

kinetic_spectrum_matrix

`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.kinetic_spectrum_matrix`

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.**KineticSpectrumModel**

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

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

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:* *Pa-*
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,* *parame-*
ters: *ParameterGroup,*
clp_labels: *list[str],*
matrix: *np.ndarray,*
index: *float) → tu-*
ple[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_concentr*

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_descriptor*)

set_initial_concentration

`KineticSpectrumModel.set_initial_concentration` (*label: str, item: glotaran.builtin.models.kinetic_image.initial_concentration_descriptor*)

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*) \rightarrow `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`) \rightarrow `np.ndarray`

constrain_matrix_function (*dataset:* `str`, *parameters:* `ParameterGroup`, *clp_labels:* `list[str]`, *matrix:* `np.ndarray`, *index:* `float`) \rightarrow `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`) \rightarrow `glotaran.model.base_model.Model`

Creates a model from a dictionary.

Parameters `model_dict` – Dictionary containing the model.

get_dataset (*label:* `str`) \rightarrow `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor`

get_initial_concentration (*label:* `str`) \rightarrow `glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration`

get_irf (*label:* `str`) \rightarrow `glotaran.builtin.models.kinetic_image.irf.Irf`

get_k_matrix (*label:* `str`) \rightarrow `glotaran.builtin.models.kinetic_image.k_matrix.KMatrix`

get_megacomplex (*label:* `str`) \rightarrow `glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.KineticImageMegacomplex`

get_shape (*label:* `str`) \rightarrow `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 () \rightarrow `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_spect**

spectral_constraints

This package contains compartment constraint items.

Functions

Summary

<i>apply_spectral_constraints</i>

apply_spectral_constraints

glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**apply_spectral_constraints**

Classes

Summary

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

<i>applies</i>	Returns true if the indexx is in one of the intervals.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>mprint</i>	
<i>validate</i>	

applies

`ZeroConstraint.applies(index: Any) → bool`

Returns true if the index 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 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]
```

spectral_irf

Classes

Summary

<i>IrfGaussianCoherentArtifact</i>	
<i>IrfSpectralGaussian</i>	
<i>IrfSpectralMultiGaussian</i>	Represents a gaussian IRF.

IrfGaussianCoherentArtifact

```
class glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussianCoherentArtifact
    Bases: glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian
```

Attributes Summary

<i>backsweep</i>
<i>backsweep_period</i>
<i>center</i>
<i>center_dispersion</i>
<i>coherent_artifact_order</i>
<i>coherent_artifact_width</i>
<i>dispersion_center</i>
<i>label</i>

continues on next page

Table 82 – continued from previous page

<i>model_dispersion_with_wavenumber</i>
<i>normalize</i>
<i>scale</i>
<i>type</i>
<i>width</i>
<i>width_dispersion</i>

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

<code>fill</code>	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.
<code>from_dict</code>	
<code>from_list</code>	
<code>mprint</code>	
<code>parameter</code>	
<code>validate</code>	

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

<i>calculate</i>	
<hr/>	
<i>calculate_dispersion</i>	
<hr/>	
<i>fill</i>	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.
<hr/>	
<i>from_dict</i>	
<hr/>	
<i>from_list</i>	
<hr/>	
<i>mprint</i>	
<hr/>	
<i>parameter</i>	
<hr/>	
<i>validate</i>	
<hr/>	

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

<i>backsweep</i>
<i>backsweep_period</i>
<i>center</i>
<i>center_dispersion</i>
<i>dispersion_center</i>
<i>label</i>
<i>model_dispersion_with_wavenumber</i>
<i>normalize</i>
<i>scale</i>
<i>type</i>
<i>width</i>
<i>width_dispersion</i>

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

```
TrfSpectralMultiGaussian.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`

property `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

<i>EqualAreaPenalty</i>	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

<i>parameter</i>
<i>source</i>
<i>source_intervals</i>
<i>target</i>
<i>target_intervals</i>
<i>weight</i>

parameter`EqualAreaPenalty.parameter`**source**`EqualAreaPenalty.source`**source_intervals**`EqualAreaPenalty.source_intervals`**target**`EqualAreaPenalty.target`**target_intervals**`EqualAreaPenalty.target_intervals`**weight**`EqualAreaPenalty.weight`**Methods Summary**

<i>applies</i>	Returns true if the index is in one of the intervals.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>mprint</i>	
<i>validate</i>	

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
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

create_spectral_relation_matrix

`glotaran.builtin.models.kinetic_spectrum.spectral_relations.create_spectral_relation_m`

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

<i>SpectralRelation</i>

SpectralRelation

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

Attributes Summary

<i>compartment</i>
<i>interval</i>
<i>parameter</i>
<i>target</i>

compartment

SpectralRelation.**compartment**

interval

SpectralRelation.**interval**

parameter

SpectralRelation.**parameter**

target

SpectralRelation.**target**

Methods Summary

<i>applies</i>	Returns true if the index is in one of the intervals.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	

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

<i>SpectralShape</i>	Base class for spectral shapes
<i>SpectralShapeGaussian</i>	A gaussian spectral shape
<i>SpectralShapeOne</i>	A gaussian spectral shape
<i>SpectralShapeZero</i>	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

<i>calculate</i>	calculate calculates the shape.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	
<i>mprint</i>	
<i>validate</i>	

calculate

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

Parameters **axis** (`np.ndarray`) – The axes 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 axes 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

<i>label</i>
<i>type</i>

label

SpectralShapeOne.**label**

type

SpectralShapeOne.**type**

Methods Summary

<i>calculate</i>	calculate calculates the shape.
<i>fill</i>	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.
<i>from_dict</i>	
<i>from_list</i>	

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 axes 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 axes 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

<code>calculate</code>	calculate calculates the shape.
<code>fill</code>	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.
<code>from_dict</code>	
<code>from_list</code>	
<code>mprint</code>	
<code>validate</code>	

calculate

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

Only works after calling calling ‘fill’.

Parameters `axis` (`np.ndarray`) – The axes 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 axes 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

<code>export</code>	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

<code>optimize_cmd</code>	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

<code>plugin_list_cmd</code>	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

<code>print_cmd</code>	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

<code>load_dataset_file</code>	
--------------------------------	--

continues on next page

Table 109 – continued from previous page

<code>load_model_file</code>
<code>load_parameter_file</code>
<code>load_scheme_file</code>
<code>select_data</code>
<code>select_name</code>
<code>signature_analysis</code>
<code>write_data</code>

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

<i>envvar_list_splitter</i>	if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up.
<i>is_composite</i>	
<i>name</i>	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

<code>convert</code>	Converts the value.
<code>fail</code>	Helper method to fail with an invalid value message.
<code>get_metavar</code>	Returns the metavar default for this param if it provides one.
<code>get_missing_message</code>	Optionally might return extra information about a missing parameter.
<code>split_envvar_value</code>	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

<i>allow_extra_args</i>	the default for the Context. allow_extra_args flag.
<i>allow_interspersed_args</i>	the default for the Context. allow_interspersed_args flag.
<i>ignore_unknown_options</i>	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

<code>add_command</code>	Registers another <code>Command</code> with this group.
<code>collect_usage_pieces</code>	Returns all the pieces that go into the usage line and returns it as a list of strings.
<code>command</code>	Behaves the same as <code>click.Group.command()</code> except capture a priority for listing command names in help.
<code>format_commands</code>	Extra format methods for multi methods that adds all the commands after the options.
<code>format_epilog</code>	Writes the epilog into the formatter if it exists.
<code>format_help</code>	Writes the help into the formatter if it exists.
<code>format_help_text</code>	Writes the help text to the formatter if it exists.
<code>format_options</code>	Writes all the options into the formatter if they exist.
<code>format_usage</code>	Writes the usage line into the formatter.
<code>get_command</code>	Given a context and a command name, this returns a <code>Command</code> object if it exists or returns <code>None</code> .
<code>get_help</code>	Formats the help into a string and returns it.
<code>get_help_option</code>	Returns the help option object.
<code>get_help_option_names</code>	Returns the names for the help option.
<code>get_params</code>	
<code>get_short_help_str</code>	Gets short help for the command or makes it by shortening the long help string.
<code>get_usage</code>	Formats the usage line into a string and returns it.
<code>group</code>	A shortcut decorator for declaring and attaching a group to the group.
<code>invoke</code>	Given a context, this invokes the attached callback (if it exists) in the right way.
<code>list_commands</code>	Returns a list of subcommand names in the order they should appear.
<code>list_commands_for_help</code>	reorder the list of commands when listing the help
<code>main</code>	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

<code>make_context</code>	This function when given an info name and arguments will kick off the parsing and create a new <code>Context</code> .
<code>make_parser</code>	Creates the underlying option parser for this command.
<code>parse_args</code>	Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary.
<code>resolve_command</code>	
<code>resultcallback</code>	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 invocation. 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'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 invocation. 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'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

<code>prepare_time_trace_dataset</code>	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,
xar-
ray.core.dataset.Dataset],
weight: *Op-*
tional[numpy.ndarray]
= None, irf: *Op-*
tional[Union[numpy.ndarray,
xar-
ray.core.dataarray.DataArray]]
= None) → xar-
ray.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

<code>glotaran.model.attribute</code>	The model attribute decorator.
<code>glotaran.model.base_model</code>	A base class for global analysis models.
<code>glotaran.model.dataset_descriptor</code>	The DatasetDescriptor class.
<code>glotaran.model.decorator</code>	The model decorator.
<code>glotaran.model.property</code>	The model property class.
<code>glotaran.model.util</code>	Helper functions.
<code>glotaran.model.weight</code>	The Weight property class.

attribute

The model attribute decorator.

Functions

Summary

<code>model_attribute</code>	The <code>@model_attribute</code> decorator adds the given properties to the class.
<code>model_attribute_typed</code>	The <code>model_attribute_typed</code> 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

<i>Model</i>	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

<i>index_dependent_matrix</i>	
<i>model_type</i>	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

<code>from_dict</code>	Creates a model from a dictionary.
<code>markdown</code>	Formats the model as Markdown string.
<code>problem_list</code>	Returns a list with all problems in the model and missing parameters if specified.
<code>simulate</code>	Simulates the model.
<code>valid</code>	Returns <i>True</i> if the number problems in the model is 0, else <i>False</i>
<code>validate</code>	Returns a string listing all problems in the model and missing parameters if specified.

from_dict

classmethod `Model.from_dict` (*model_dict_ref*: *dict*) \rightarrow
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*)
 \rightarrow *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 descriptor 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:      Op-
                                tional[Dict[str, Any]] = None, dataset_type:
                                Type[glotaran.model.dataset_descriptor.DatasetDescriptor]
                                = <class 'glotaran.model.dataset_descriptor.DatasetDescriptor'>,
                                megacomplex_type:      Op-
                                tional[Any] = None, matrix:      Op-
                                tional[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: Op-
                                tional[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: Op-
                                tional[Callable[[Type[glotaran.model.base_model.Model],
                                bool]] = None, constrain_matrix_function: Op-
                                tional[Callable[[Type[glotaran.model.base_model.Model],
                                glotaran.parameter.parameter_group.ParameterGroup,
                                List[str], numpy.ndarray, float], Tu-
                                ple[List[str], numpy.ndarray]]] =
                                None, retrieve_clp_function: Op-
                                tional[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: Op-
                                tional[Callable[[Type[glotaran.model.base_model.Model],
                                bool]] = None, additional_penalty_function: Op-
                                tional[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, xar-
                                ray.core.dataset.Dataset], float], numpy.ndarray]]
                                = None, finalize_data_function: Op-
                                tional[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

<i>deleter</i>	Descriptor to change the deleter on a property.
<i>fill</i>	
<i>getter</i>	Descriptor to change the getter on a property.
<i>setter</i>	Descriptor to change the setter on a property.
<i>validate</i>	

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**

<code>wrap_func_as_method</code>	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

<i>Weight</i>	The <i>Weight</i> 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

<i>datasets</i>
<i>global_interval</i>
<i>model_interval</i>
<i>value</i>

datasets`Weight.datasets`**global_interval**`Weight.global_interval`**model_interval**`Weight.model_interval`**value**`Weight.value`**Methods Summary**

<code>fill</code>	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.
<code>from_dict</code>	
<code>from_list</code>	
<code>mprint</code>	
<code>validate</code>	

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**

<code>glotaran.parameter.parameter</code>	The parameter class.
<code>glotaran.parameter.parameter_group</code>	The parameter group class

parameter

The parameter class.

Classes

Summary

<i>Keys</i>	Keys for parameter options.
<i>Parameter</i>	A parameter for optimization.

Keys

class `glotaran.parameter.parameter.Keys`
Bases: `object`
Keys for parameter options.

Attributes Summary

<i>EXPR</i>
<i>MAX</i>
<i>MIN</i>
<i>NON_NEG</i>
<i>VARY</i>

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

<i>expression</i>	The expression of the parameter.
<i>full_label</i>	The label of the parameter with its path in a parameter group prepended.
<i>label</i>	Label of the parameter
<i>maximum</i>	The upper bound of the parameter.
<i>minimum</i>	The lower bound of the parameter.
<i>non_negative</i>	Indicates if the parameter is non-negativ.
<i>standard_error</i>	The standard error of the optimized parameter.
<i>transformed_expression</i>	The expression of the parameter transformed for evaluation within a <i>ParameterGroup</i> .
<i>value</i>	The value of the parameter
<i>vary</i>	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-negative.

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**

<code>from_list_or_value</code>	Creates a parameter from a list or numeric value.
<code>get_value_and_bounds_for_optimization</code>	Gets the parameter value and bounds with expression and non-negative constraints applied.
<code>set_from_group</code>	Sets all values of the parameter to the values of the corresponding parameter in the group.
<code>set_value_from_optimization</code>	Sets the value from an optimization result and reverses non-negative transformation.
<code>valid_label</code>	Returns true if the <i>label</i> 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() \rightarrow 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-negative.

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) \rightarrow 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

<i>ParameterGroup</i>	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

<i>label</i>	Label of the group.
<i>root_group</i>	Root of the group.

label

`ParameterGroup.label`
Label of the group.

root_group

`ParameterGroup.root_group`
Root of the group.

Methods Summary

<code>add_group</code>	Adds a <i>ParameterGroup</i> to the group.
<code>add_parameter</code>	Adds a <i>Parameter</i> to the group.
<code>all</code>	Returns a generator over all parameter in the group and it's subgroups together with their labels.
<code>clear</code>	
<code>copy</code>	
<code>from_csv</code>	Creates a <i>ParameterGroup</i> from a CSV file.
<code>from_dataframe</code>	Creates a <i>ParameterGroup</i> from a <code>pandas.DataFrame</code>
<code>from_dict</code>	Creates a <i>ParameterGroup</i> from a dictionary.
<code>from_file</code>	
<code>from_list</code>	Creates a <i>ParameterGroup</i> from a list.
<code>from_yaml</code>	Creates a <i>ParameterGroup</i> from a YAML string.
<code>from_yaml_file</code>	Creates a <i>ParameterGroup</i> from a YAML file.
<code>fromkeys</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get</code>	Gets a <i>Parameter</i> by its label.
<code>get_label_value_and_bounds_arrays</code>	Returns a arrays of all parameter labels, values and bounds.
<code>get_nr_roots</code>	Returns the number of roots of the group.
<code>groups</code>	Returns a generator over all groups and their subgroups.
<code>has</code>	Checks if a parameter with the given label is in the group or in a subgroup.
<code>items</code>	
<code>keys</code>	
<code>known_formats</code>	

continues on next page

Table 147 – continued from previous page

<code>markdown</code>	Formats the <i>ParameterGroup</i> as mark-down string.
<code>pop</code>	If key is not found, d is returned if given, otherwise <code>KeyError</code> is raised
<code>popitem</code>	Remove and return a (key, value) pair as a 2-tuple.
<code>set_from_label_and_value_arrays</code>	Updates the parameter values from a list of labels and values.
<code>setdefault</code>	Insert key with a value of default if key is not in the dictionary.
<code>to_csv</code>	Writes a <i>ParameterGroup</i> to a CSV file.
<code>to_dataframe</code>	
<code>update</code>	If E is present and has a <code>.keys()</code> method, then does: for k in E: D[k] = E[k] If E is present and lacks a <code>.keys()</code> 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]
<code>update_parameter_expression</code>	Updates all parameters which have an expression.
<code>values</code>	

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

<code>glotaran.parse.parser</code>	Functions for reading and parsing models from serialized representations.
<code>glotaran.parse.register</code>	A register for models
<code>glotaran.parse.util</code>	

parser

Functions for reading and parsing models from serialized representations.

Functions

Summary

<code>load_yaml</code>	
<code>load_yaml_file</code>	
<code>parse_spec</code>	
<code>parse_yaml</code>	
<code>parse_yaml_file</code>	<code>parse_yaml_file</code> reads the given file and parses its content as YAML.

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 YAML.

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

<code>get_model</code>	<code>get_model</code> gets a model from the register.
<code>known_model</code>	<code>known_model</code> returns True if the <code>model_type</code> is in the register.
<code>known_model_names</code>	
<code>register_model</code>	<code>register_model</code> 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

<code>list_string_to_tuple</code>	Converts a list of strings (representing tuples) to a list of tuples
<code>sanitize_dict_keys</code>	Sanitize the stringified tuple dict keys in a yaml parsed dict
<code>sanitize_dict_values</code>	Sanitizes a dict with broken tuples inside modifying it in-place Broken tuples are tuples that are turned into strings by the yaml parser.
<code>sanitize_list_with_broken_tuples</code>	Sanitize a list with 'broken' tuples
<code>sanitize_yaml</code>	Sanitize a yaml-returned dict for key or (list) values containing tuples
<code>string_to_tuple</code>	[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 repressed 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 function 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 restored 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 yaml 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]]

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/glottaran/pyglottaran/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

pyglottaran could always use more documentation, whether as part of the official pyglottaran 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 and 3.9 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), the version number only needs to be changed in `glotaran/__init__.py`.

Then make a [new release on GitHub](#) and give the tag a proper name, e.g. `0.3.0` since might be included in a citation. Github Actions will then deploy to PyPI if the tests pass.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [1] https://glotaran.github.io/legacy/file_formats

PYTHON MODULE INDEX

g

glotaran, 29
 glotaran.analysis, 29
 glotaran.analysis.nnls, 30
 glotaran.analysis.optimize, 31
 glotaran.analysis.problem, 31
 glotaran.analysis.result, 47
 glotaran.analysis.scheme, 54
 glotaran.analysis.simulation, 58
 glotaran.analysis.variable_projection, 59
 glotaran.builtin, 59
 glotaran.builtin.file_formats, 60
 glotaran.builtin.file_formats.ascii, 60
 glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file, 60
 glotaran.builtin.file_formats.sdt, 69
 glotaran.builtin.file_formats.sdt.sdt_file_reader, 70
 glotaran.builtin.models, 71
 glotaran.builtin.models.kinetic_image, 71
 glotaran.builtin.models.kinetic_image.initial_concentration, 72
 glotaran.builtin.models.kinetic_image.irf, 75
 glotaran.builtin.models.kinetic_image.k_matrix, 84
 glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor, 90
 glotaran.builtin.models.kinetic_image.kinetic_image_matrix, 94
 glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex, 95
 glotaran.builtin.models.kinetic_image.kinetic_image_model, 98
 glotaran.builtin.models.kinetic_image.kinetic_image_result, 106
 glotaran.builtin.models.kinetic_spectrum, 107
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor, 108
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum, 127
 glotaran.builtin.models.kinetic_spectrum.spectral, 128
 glotaran.builtin.models.kinetic_spectrum.spectral, 134
 glotaran.builtin.models.kinetic_spectrum.spectral, 148
 glotaran.builtin.models.kinetic_spectrum.spectral, 148
 glotaran.builtin.models.kinetic_spectrum.spectral, 151
 glotaran.builtin.models.kinetic_spectrum.spectral, 157
 glotaran.cli, 165
 glotaran.cli.commands, 166
 glotaran.cli.commands.explore, 166
 glotaran.cli.commands.export, 166
 glotaran.cli.commands.optimize, 166
 glotaran.cli.commands.pluginlist, 167
 glotaran.cli.commands.print, 167
 glotaran.cli.commands.util, 167
 glotaran.cli.commands.validate, 171
 glotaran.cli.main, 172
 glotaran.examples, 181
 glotaran.io.prepare_dataset, 182
 glotaran.io.reader, 183
 glotaran.model, 184
 glotaran.model.attribute, 184
 glotaran.model.base_model, 185
 glotaran.model.dataset_descriptor, 188
 glotaran.model.decorator, 191
 glotaran.model.property, 194
 glotaran.model.util, 196
 glotaran.model.weight, 197
 glotaran.parameter, 199

`glotaran.parameter.parameter`, [200](#)
`glotaran.parameter.parameter_group`, [205](#)
`glotaran.parse`, [215](#)
`glotaran.parse.parser`, [216](#)
`glotaran.parse.register`, [217](#)
`glotaran.parse.util`, [218](#)

A

`a_matrix()` (`glotaran.builtin.models.kinetic_image.k_matrix.KMatrix` method), 89
`a_matrix_as_markdown()` (`glotaran.builtin.models.kinetic_image.k_matrix.KMatrix` method), 89
`a_matrix_non_unibranch()` (`glotaran.builtin.models.kinetic_image.k_matrix.KMatrix` method), 89
`a_matrix_unibranch()` (`glotaran.builtin.models.kinetic_image.k_matrix.KMatrix` method), 89
`add_command()` (`glotaran.cli.main.Cli` method), 178
`add_data_row()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile` method), 66
`add_data_row()` (`glotaran.builtin.file_formats.ascii.wavelength_explicit_file.WavelengthExplicitFile` method), 69
`add_equal_area_penalties()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`add_group()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 213
`add_parameter()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 213
`add_spectral_constraints()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`add_spectral_relations()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`add_type()` (`glotaran.builtin.models.kinetic_image.irf.Irf` class method), 75
`add_type()` (`glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint` class method), 131
`add_type()` (`glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape` class method), 158
`add_weights()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`add_weights()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`additional_penalty()` (`glotaran.analysis.problem.Problem` property), 43
`additional_penalty()` (`glotaran.analysis.result.Result` property), 53
`additional_penalty_function` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` attribute), 105
`additional_penalty_function()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`all()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 213
`allow_extra_args` (`glotaran.cli.main.Cli` attribute), 178
`allow_interspersed_args` (`glotaran.cli.main.Cli` attribute), 178
`allow_none()` (`glotaran.model.property.ModelProperty` property), 196
`amplitude()` (`glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape` property), 160
`applies()` (`glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint` method), 130
`applies()` (`glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint` method), 133
`applies()` (`glotaran.builtin.models.kinetic_spectrum.spectral_penalties.SpectralPenalty` method), 150
`applies()` (`glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelation` method), 157
`apply_kinetic_model_constraints()` (in module `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel`), 112
`apply_spectral_constraints()` (in module `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel`), 128
`apply_spectral_penalties()` (in module `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel`), 113
`apply_spectral_relations()` (in module `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel`), 152
`axis` (`glotaran.analysis.problem.GroupedProblemDescriptor` attribute), 35

B

`calculate_coherent_artifact()`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian method), 138`
`calculate_dispersion()`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian method), 138`
`calculate_index_dependent_grouped_matrices()`
`(glotaran.analysis.problem.Problem method), 43`
`calculate_index_dependent_grouped_residual()`
`(glotaran.analysis.problem.Problem method), 43`
`calculate_index_dependent_ungrouped_matrices()`
`(glotaran.analysis.problem.Problem method), 43`
`calculate_index_dependent_ungrouped_residual()`
`(glotaran.analysis.problem.Problem method), 43`
`calculate_index_independent_grouped_matrices()`
`(glotaran.analysis.problem.Problem method), 44`
`calculate_index_independent_grouped_residual()`
`(glotaran.analysis.problem.Problem method), 44`
`calculate_index_independent_ungrouped_residual()`
`(glotaran.analysis.problem.Problem method), 44`
`calculate_kinetic_matrix_gaussian_irf()`
`(in module glotaran.builtin.models.kinetic_image.kinetic_image_mat_1 method), 84`
`calculate_kinetic_matrix_no_irf()`
`(in module glotaran.builtin.models.kinetic_image.kinetic_image_mat_1 method), 94`
`calculate_matrices()`
`(glotaran.analysis.problem.Problem method), 147`
`calculate_residual()`
`(glotaran.analysis.problem.Problem method), 160`
`callback (glotaran.cli.main.Cli attribute), 178`
`center()`
`(glotaran.builtin.models.kinetic_image.irf.IrfGaussian property), 78`
`(glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian property), 84`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 138`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian property), 142`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeGaussian property), 160`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeOne property), 163`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeZero property), 165`
`calculate_additional_penalty()`
`(glotaran.analysis.problem.Problem method), 43`
`backsweep()`
`(glotaran.builtin.models.kinetic_image.irf.IrfGaussian property), 78`
`(glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian property), 84`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 138`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian property), 142`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147`
`backsweep_period()`
`(glotaran.builtin.models.kinetic_image.irf.IrfGaussian property), 78`
`(glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian property), 84`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 138`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian property), 142`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147`
`bag()`
`(glotaran.analysis.problem.Problem property), 43`
`baseline()`
`(glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor property), 93`
`(glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor property), 111`

C

`calculate_index_independent_ungrouped_residual()`
`(glotaran.analysis.problem.Problem method), 44`
`calculate_kinetic_matrix_gaussian_irf()`
`(in module glotaran.builtin.models.kinetic_image.kinetic_image_mat_1 method), 84`
`calculate_kinetic_matrix_no_irf()`
`(in module glotaran.builtin.models.kinetic_image.kinetic_image_mat_1 method), 94`
`calculate_matrices()`
`(glotaran.analysis.problem.Problem method), 147`
`calculate_residual()`
`(glotaran.analysis.problem.Problem method), 160`
`callback (glotaran.cli.main.Cli attribute), 178`
`center()`
`(glotaran.builtin.models.kinetic_image.irf.IrfGaussian property), 78`
`(glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian property), 84`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 138`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian property), 142`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeGaussian property), 160`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeOne property), 163`
`(glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralShapeZero property), 165`
`calculate_additional_penalty()`
`(glotaran.analysis.problem.Problem method), 43`

[illegible]

D

DatasetDescriptor (class in file_reader() (in module glotaran.io.reader), 183
 glotaran.model.dataset_descriptor), 189
 datasets() (glotaran.model.weight.Weight property), 199
 degrees_of_freedom() (glotaran.analysis.result.Result property), 53
 deleter() (glotaran.model.property.ModelProperty method), 196
 descriptor (glotaran.analysis.problem.GroupedProblem attribute), 33
 dispersion_center() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 138
 dispersion_center() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian property), 142
 dispersion_center() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147
E
 eigen() (glotaran.builtin.models.kinetic_image.k_matrix.KMatrix method), 89
 empty() (glotaran.builtin.models.kinetic_image.k_matrix.KMatrix class method), 89
 envvar_list_splitter (glotaran.cli.commands.util.ValOrRangeOrList attribute), 171
 equal_area_penalties() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property), 125
 EqualAreaPenalty (class in glotaran.builtin.models.kinetic_spectrum.spectral_penalties.SpectralPenalties), 148
 exclude_from_normalize() (glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration property), 74
 ExplicitFile (class in glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file), (glotaran.model.property.ModelProperty method), 196
 export() (in module glotaran.cli.commands.explore), 166
 EXPR (glotaran.parameter.parameter.Keys attribute), 201
 expression() (glotaran.parameter.parameter.Parameter property), 204
F
 fail() (glotaran.cli.commands.util.ValOrRangeOrList method), 171
 fdel (glotaran.model.property.ModelProperty attribute), 196
 fget (glotaran.model.property.ModelProperty attribute), 196
 fill() (glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration method), 74
 fill() (glotaran.builtin.models.kinetic_image.irf.IrfGaussian method), 78
 fill() (glotaran.builtin.models.kinetic_image.irf.IrfMeasured method), 80
 fill() (glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian method), 84
 fill() (glotaran.builtin.models.kinetic_image.k_matrix.KMatrix method), 89
 fill() (glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor method), 93
 fill() (glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex.MegaComplex method), 97
 fill() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor method), 111
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_constraints.OrbitalConstraints method), 133
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian method), 138
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian method), 142
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian method), 147
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty method), 150
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelations method), 157
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape method), 163
 fill() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape method), 165
 fill() (glotaran.model.dataset_descriptor.DatasetDescriptor method), 190
 fill() (glotaran.model.weight.Weight method), 199
 filled_dataset_descriptors() (glotaran.analysis.problem.Problem property), 44
 finalize_data() (glotaran.builtin.models.kinetic_image.kinetic_image_result.KineticImageResult method), 105
 finalize_data() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result.KineticSpectrumResult method), 125
 finalize_kinetic_image_result() (in module glotaran.builtin.models.kinetic_image.kinetic_image_result), 107
 finalize_kinetic_spectrum_result() (in module glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result), 107

[illegible]

`from_list()` (`glotaran.parameter.parameter_group.ParameterGroup` class method), 213
`from_list_or_value()` (`glotaran.parameter.parameter.Parameter` class method), 204
`from_yaml()` (`glotaran.parameter.parameter_group.ParameterGroup` class method), 213
`from_yaml_file()` (`glotaran.analysis.scheme.Scheme` class method), 58
`from_yaml_file()` (`glotaran.parameter.parameter_group.ParameterGroup` class method), 213
`fromkeys()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
`fset` (`glotaran.model.property.ModelProperty` attribute), 196
`ftol()` (`glotaran.analysis.scheme.Scheme` property), 58
`full()` (`glotaran.builtin.models.kinetic_image.k_matrix.KMatrix` method), 90
`full_k_matrix()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 97
`full_label()` (`glotaran.parameter.parameter.Parameter` property), 204
`full_penalty()` (`glotaran.analysis.problem.Problem` property), 44
G
`get()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
`get_command()` (`glotaran.cli.main.Cli` method), 179
`get_data_file_format()` (in module `glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file`), 60
`get_data_row()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile` method), 64
`get_data_row()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile` method), 66
`get_data_row()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile` method), 69
`get_dataset()` (`glotaran.analysis.result.Result` method), 53
`get_dataset()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`get_dataset()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`get_explicit_axis()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile` method), 64
`get_explicit_axis()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile` method), 66
`get_explicit_axis()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile` method), 69
`get_format_name()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file` method), 64
`get_format_name()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file` method), 66
`get_format_name()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file` method), 69
`get_help_option()` (`glotaran.cli.main.Cli` method), 179
`get_help_option_names()` (`glotaran.cli.main.Cli` method), 179
`get_initial_concentration()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`get_initial_concentration()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`get_interval_number()` (in module `glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file`), 61
`get_irf()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`get_irf()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`get_k_matrices()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 93
`get_k_matrices()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 111
`get_k_matrix()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`get_k_matrix()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`get_label_value_and_bounds_arrays()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
`get_megacomplex()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 105
`get_megacomplex()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 125
`get_metavar()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 171
`get_missing_message()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 171
`get_model()` (in module `glotaran.parse.register`), 217
`get_nr_roots()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
`get_observations()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file` method), 69

method), 64
 get_observations() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile*
method), 66
 get_observations() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile*
method), 69
 get_params() (*glotaran.cli.main.Cli method*), 179
 get_scheme() (*glotaran.analysis.result.Result*
method), 53
 get_secondary_axis() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile*
method), 64
 get_secondary_axis() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile*
method), 66
 get_secondary_axis() (*glotaran.builtin.file_formats.sdt.sdt_file_reader*
method), 70
 get_shape() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel*
method), 125
 get_short_help_str() (*glotaran.cli.main.Cli*
method), 179
 get_usage() (*glotaran.cli.main.Cli method*), 179
 get_value_and_bounds_for_optimization() (*glotaran.parameter.parameter.Parameter*
method), 205
 getter() (*glotaran.model.property.ModelProperty*
method), 196
 global_axis (*glotaran.analysis.problem.ProblemDescription*
attribute), 47
 global_dimension (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel*
attribute), 105
 global_dimension (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel*
attribute), 125
 global_interval() (*glotaran.model.weight.Weight*
property), 199
 global_matrix (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel*
attribute), 105
 global_matrix() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel*
static method), 125
 glotaran
 module, 29
 glotaran.analysis
 module, 29
 glotaran.analysis.nnls
 module, 30
 glotaran.analysis.optimize
 module, 31
 glotaran.analysis.problem
 module, 31
 glotaran.analysis.result
 module, 47
 glotaran.analysis.scheme
 module, 54
 glotaran.analysis.simulation
 glotaran.analysis.variable_projection
 module, 59
 module, 59
 glotaran.builtin.file_formats
 module, 60
 glotaran.builtin.file_formats.ascii
 module, 60
 module, 60
 glotaran.builtin.file_formats.sdt
 glotaran.builtin.file_formats.sdt.sdt_file_reader
 module, 70
 module, 71
 glotaran.builtin.models.kinetic_image
 module, 71
 glotaran.builtin.models.kinetic_image.initial_conc
 module, 72
 glotaran.builtin.models.kinetic_image.irf
 module, 75
 glotaran.builtin.models.kinetic_image.k_matrix
 module, 84
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 90
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 94
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 95
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 98
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 106
 glotaran.builtin.models.kinetic_image.kinetic_image
 module, 107
 glotaran.builtin.models.kinetic_spectrum
 module, 108
 glotaran.builtin.models.kinetic_spectrum.kinetic_sp
 module, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_sp
 module, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_sp
 module, 127
 glotaran.builtin.models.kinetic_spectrum.spectral_c
 module, 128
 glotaran.builtin.models.kinetic_spectrum.spectral_
 module, 134
 glotaran.builtin.models.kinetic_spectrum.spectral_r
 module, 148
 glotaran.builtin.models.kinetic_spectrum.spectral_p

- module, 148
- glotaran.builtin.models.kinetic_spectrum_group
 - module, 151
- glotaran.builtin.models.kinetic_spectrum_group_spectral_shape
 - module, 157
- glotaran.cli
 - module, 165
- glotaran.cli.commands
 - module, 166
- glotaran.cli.commands.explore
 - module, 166
- glotaran.cli.commands.export
 - module, 166
- glotaran.cli.commands.optimize
 - module, 166
- glotaran.cli.commands.pluginlist
 - module, 167
- glotaran.cli.commands.print
 - module, 167
- glotaran.cli.commands.util
 - module, 167
- glotaran.cli.commands.validate
 - module, 171
- glotaran.cli.main
 - module, 172
- glotaran.examples
 - module, 181
- glotaran.examples.sequential
 - module, 182
- glotaran.io
 - module, 182
- glotaran.io.prepare_dataset
 - module, 182
- glotaran.io.reader
 - module, 183
- glotaran.model
 - module, 184
- glotaran.model.attribute
 - module, 184
- glotaran.model.base_model
 - module, 185
- glotaran.model.dataset_descriptor
 - module, 188
- glotaran.model.decorator
 - module, 191
- glotaran.model.property
 - module, 194
- glotaran.model.util
 - module, 196
- glotaran.model.weight
 - module, 197
- glotaran.parameter
 - module, 199
- glotaran.parameter.parameter

- module, 200
- glotaran.parameter.parameter_group
 - module, 205
- glotaran.parameter_shape
 - module, 215
- glotaran.parse.parser
 - module, 216
- glotaran.parse.register
 - module, 217
- glotaran.parse.util
 - module, 218
- group (*glotaran.analysis.problem.GroupedProblem* attribute), 33
- group() (*glotaran.cli.main.Cli* method), 180
- group_tolerance()
 - (*glotaran.analysis.scheme.Scheme* property), 58
- grouped() (*glotaran.analysis.problem.Problem* property), 44
- grouped() (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel* method), 105
- grouped() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel* method), 125
- grouped()
 - (in *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel* module), 114
- GroupedProblem
 - (class in *glotaran.analysis.problem*), 32
- GroupedProblemDescriptor
 - (class in *glotaran.analysis.problem*), 34
- groups() (*glotaran.analysis.problem.Problem* property), 44
- groups() (*glotaran.parameter.parameter_group.ParameterGroup* method), 214
- gtol() (*glotaran.analysis.scheme.Scheme* property), 58

H

- has() (*glotaran.parameter.parameter_group.ParameterGroup* method), 214
- has_additional_penalty_function
 - (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel* attribute), 105
- has_additional_penalty_function()
 - (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel* method), 125
- has_kinetic_model_constraints() (in module *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel*), 114
- has_matrix_constraints_function
 - (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel* attribute), 105
- has_matrix_constraints_function()
 - (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel* method), 125

has_scaling(*glotaran.analysis.problem.GroupedProblem attribute*), 33
 has_spectral_penalties() (in module *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model*), 131
 114
 I
 ignore_unknown_options(*glotaran.cli.main.Cli attribute*), 180
 index(*glotaran.analysis.problem.GroupedProblemDescriptor attribute*), 35
 index() (*glotaran.analysis.problem.GroupedProblem method*), 33
 index() (*glotaran.analysis.problem.LabelAndMatrix method*), 36
 index() (*glotaran.analysis.problem.ProblemDescriptor method*), 47
 index_dependent() (*glotaran.analysis.problem.Problem property*), 44
 index_dependent() (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel method*), 105
 index_dependent() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel method*), 126
 index_dependent() (in module *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model*), 114
 index_dependent_matrix() (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
 index_dependent_matrix() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
 index_dependent_matrix() (*glotaran.model.base_model.Model property*), 187
 initial_concentration() (*glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor property*), 93
 initial_concentration() (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
 initial_concentration() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor property*), 111
 initial_concentration() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
 initial_parameters() (*glotaran.analysis.result.Result property*), 53
 InitialConcentration (class in *glotaran.builtin.models.kinetic_image.initial_concentration*), 72
 interval() (*glotaran.builtin.models.kinetic_spectrum.spectral_constraint.Interval property*), 131
 interval() (*glotaran.builtin.models.kinetic_spectrum.spectral_constraint.Interval property*), 134
 interval() (*glotaran.builtin.models.kinetic_spectrum.spectral_constraint.Interval property*), 157
 invoke() (*glotaran.cli.main.Cli method*), 180
 involved_compartments() (*glotaran.builtin.models.kinetic_image.k_matrix.KMatrix method*), 90
 involved_compartments() (*glotaran.builtin.models.kinetic_image.kinetic_image_megacompartments.Megacompartments property*), 97
 Irf (class in *glotaran.builtin.models.kinetic_image.irf*), 75
 irf() (*glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor property*), 93
 irf() (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
 irf() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor property*), 111
 irf() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
 IrfGaussian (class in *glotaran.builtin.models.kinetic_image.irf*), 75
 IrfGaussianCoherentArtifact (class in *glotaran.builtin.models.kinetic_spectrum.spectral_irf*), 124
 IrfMeasured (class in *glotaran.builtin.models.kinetic_image.irf*), 79
 IrfMultiGaussian (class in *glotaran.builtin.models.kinetic_image.irf*), 81
 IrfSpectralGaussian (class in *glotaran.builtin.models.kinetic_spectrum.spectral_irf*), 129
 IrfSpectralMultiGaussian (class in *glotaran.builtin.models.kinetic_spectrum.spectral_irf*), 132
 is_composite(*glotaran.cli.commands.util.ValOrRangeOrList attribute*), 171
 is_composite(*glotaran.model.base_model.Model property*), 187
 is_composite(*glotaran.model.kinetic_image_model.KineticImageModel property*), 105
 is_composite(*glotaran.model.kinetic_spectrum_model.KineticSpectrumModel property*), 126
 items() (*glotaran.parameter.parameter_group.ParameterGroup method*), 90
 J
 jacobian() (*glotaran.analysis.result.Result property*), 53

K

[label\(\)](#) (`glotaran.builtin.models.kinetic_image.irf.IrfMeasured` [property](#)), 80
[k_matrix\(\)](#) (`glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex` [property](#)), 97
[k_matrix\(\)](#) (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` [property](#)), 105
[k_matrix\(\)](#) (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` [property](#)), 126
[Keys](#) (`class in glotaran.parameter.parameter`), 200
[keys\(\)](#) (`glotaran.parameter.parameter_group.ParameterGroup` [method](#)), 214
[kinetic_image_matrix\(\)](#) (`in module glotaran.builtin.models.kinetic_image.kinetic_image_model`), 98
[kinetic_image_matrix_implementation\(\)](#) (`in module glotaran.builtin.models.kinetic_image.kinetic_image_model`), 94
[kinetic_matrix\(\)](#) (`in module glotaran.builtin.models.kinetic_image.kinetic_image_matrix`), 95
[kinetic_spectrum_matrix\(\)](#) (`in module glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model`), 114
[KineticImageDatasetDescriptor](#) (`class in glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor`), 91
[KineticImageMegaComplex](#) (`class in glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex`), 95
[KineticImageModel](#) (`class in glotaran.builtin.models.kinetic_image.kinetic_image_model`), 98
[KineticSpectrumDatasetDescriptor](#) (`class in glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor`), 108
[KineticSpectrumModel](#) (`class in glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model`), 116
[KMatrix](#) (`class in glotaran.builtin.models.kinetic_image.k_matrix`), 85
[known_formats\(\)](#) (`glotaran.parameter.parameter_group.ParameterGroup` [class method](#)), 214
[known_model\(\)](#) (`in module glotaran.parse.register`), 217
[known_model_names\(\)](#) (`in module glotaran.parse.register`), 217

L

[label](#) (`glotaran.analysis.problem.GroupedProblemDescriptor` [attribute](#)), 35
[label\(\)](#) (`glotaran.builtin.models.kinetic_image.initial_concentration` [property](#)), 74
[label\(\)](#) (`glotaran.builtin.models.kinetic_image.irf.IrfGaussian` [property](#)), 78

M

[main\(\)](#) (`glotaran.cli.main.Cli` [method](#)), 180

[make_context\(\)](#) (*glotaran.cli.main.Cli method*), 180
[make_parser\(\)](#) (*glotaran.cli.main.Cli method*), 180
[markdown\(\)](#) (*glotaran.analysis.result.Result method*), 53
[markdown\(\)](#) (*glotaran.analysis.scheme.Scheme method*), 58
[markdown\(\)](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel method*), 105
[markdown\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel method*), 126
[markdown\(\)](#) (*glotaran.model.base_model.Model method*), 187
[markdown\(\)](#) (*glotaran.parameter.parameter_group.ParameterGroup method*), 214
[matrices\(\)](#) (*glotaran.analysis.problem.Problem property*), 44
[matrix\(\)](#) (*glotaran.analysis.problem.LabelAndMatrix attribute*), 36
[matrix\(\)](#) (*glotaran.builtin.models.kinetic_image.k_matrix.KMatrix property*), 90
[matrix\(\)](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel static method*), 105
[matrix\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel static method*), 126
[matrix_as_markdown\(\)](#) (*glotaran.builtin.models.kinetic_image.k_matrix.KMatrix method*), 90
[MAX](#) (*glotaran.parameter.parameter.Keys attribute*), 201
[maximum\(\)](#) (*glotaran.parameter.parameter.Parameter property*), 205
[maximum_number_function_evaluations\(\)](#) (*glotaran.analysis.scheme.Scheme property*), 58
[megacomplex\(\)](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor property*), 93
[megacomplex\(\)](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
[megacomplex\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor property*), 111
[megacomplex\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
[megacomplex\(\)](#) (*glotaran.model.dataset_descriptor.DatasetDescriptor property*), 190
[MIN](#) (*glotaran.parameter.parameter.Keys attribute*), 201
[minimum\(\)](#) (*glotaran.parameter.parameter.Parameter property*), 205
[Model](#) (*class in glotaran.model.base_model*), 185
[model\(\)](#) (*glotaran.analysis.problem.Problem property*), 44
[model\(\)](#) (*glotaran.analysis.result.Result property*), 53
[model\(\)](#) (*glotaran.analysis.scheme.Scheme property*), 58
[model\(\)](#) (*in module glotaran.model.decorator*), 192
[model_attribute\(\)](#) (*in module glotaran.model.attribute*), 184
[model_attribute_typed\(\)](#) (*in module glotaran.model.attribute*), 185
[model_axis](#) (*glotaran.analysis.problem.ProblemDescriptor attribute*), 47
[model_dimension](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
[model_dimension](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
[model_dispersion_with_wavenumber\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property*), 139
[model_dispersion_with_wavenumber\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral property*), 143
[model_dispersion_with_wavenumber\(\)](#) (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral property*), 147
[model_interval\(\)](#) (*glotaran.model.weight.Weight property*), 199
[model_kinetic_image_model](#) (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel property*), 105
[model_kinetic_spectrum_model](#) (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel property*), 126
[model_type\(\)](#) (*glotaran.model.base_model.Model property*), 188
[ModelProperty](#) (*class in glotaran.model.property*), 194
[module](#)
[glotaran](#), 29
[glotaran.analysis](#), 29
[glotaran.analysis.nnls](#), 30
[glotaran.analysis.optimize](#), 31
[glotaran.analysis.dataset_descriptor.KineticImageDatasetDescriptor](#), 47
[glotaran.analysis.result](#), 47
[glotaran.analysis.scheme](#), 58
[glotaran.analysis.simulation](#), 58
[glotaran.analysis.problem.LabelAndMatrix](#), 36
[glotaran.analysis.problem.Problem](#), 44
[glotaran.analysis.problem.ProblemDescriptor](#), 47
[glotaran.analysis.result.Result](#), 53
[glotaran.analysis.scheme.Scheme](#), 58
[glotaran.builtin.file_formats](#), 60
[glotaran.builtin.file_formats.ascii](#), 60
[glotaran.builtin.file_formats.ascii.wavelength](#), 60
[glotaran.builtin.file_formats.sdt](#), 69
[glotaran.builtin.file_formats.sdt.sdt_file_read](#), 70
[glotaran.builtin.models](#), 71
[glotaran.builtin.models.kinetic_image](#), 71
[glotaran.builtin.models.kinetic_image.initial_c](#), 72

glotaran.builtin.models.kinetic_image.irf, 75
 glotaran.builtin.models.kinetic_image.k_matrix, 84
 glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor, 90
 glotaran.builtin.models.kinetic_image.kinetic_image_matrix_weight, 94
 glotaran.builtin.models.kinetic_image.kinetic_image_parameter_group, 95
 glotaran.builtin.models.kinetic_image.kinetic_image_model, 98
 glotaran.builtin.models.kinetic_image.kinetic_image_parser, 106
 glotaran.builtin.models.kinetic_spectrum, 107
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor, 108
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_matrix, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model, 111
 glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result, 127
 glotaran.builtin.models.kinetic_spectrum.spectral_constraints, 128
 glotaran.builtin.models.kinetic_spectrum.spectral_irf, 134
 glotaran.builtin.models.kinetic_spectrum.spectral_matrix, 148
 glotaran.builtin.models.kinetic_spectrum.spectral_penalties, 148
 glotaran.builtin.models.kinetic_spectrum.spectral_relations, 151
 glotaran.builtin.models.kinetic_spectrum.spectral_shape, 157
 glotaran.cli, 165
 glotaran.cli.commands, 166
 glotaran.cli.commands.explore, 166
 glotaran.cli.commands.export, 166
 glotaran.cli.commands.optimize, 166
 glotaran.cli.commands.pluginlist, 167
 glotaran.cli.commands.print, 167
 glotaran.cli.commands.util, 167
 glotaran.cli.commands.validate, 171
 glotaran.cli.main, 172
 glotaran.examples, 181
 glotaran.examples.sequential, 182
 glotaran.io, 182
 glotaran.io.prepare_dataset, 182
 glotaran.io.reader, 183
 glotaran.model, 184
 glotaran.model.attribute, 184
 glotaran.model.base_model, 185
 glotaran.model.dataset_descriptor, 188
 glotaran.model.decorator, 191
 glotaran.model.dataset_descriptor, 194
 glotaran.model.util, 196
 glotaran.parameter, 199
 glotaran.parameter_group, 200
 glotaran.parse, 215
 glotaran.parse.parser, 216
 glotaran.parse.register, 217
 glotaran.parse.util, 218
 mprint() (glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration method), 74
 mprint() (glotaran.builtin.models.kinetic_image.irf.IrfGaussian method), 78
 mprint() (glotaran.builtin.models.kinetic_image.irf.IrfMeasured method), 80
 mprint() (glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian method), 84
 mprint() (glotaran.builtin.models.kinetic_image.k_matrix.KMatrix method), 90
 mprint() (glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor method), 93
 mprint() (glotaran.builtin.models.kinetic_image.kinetic_image_megacon.Megacon method), 97
 mprint() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor method), 134
 mprint() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel method), 134
 mprint() (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result.KineticSpectrumResult method), 134
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraints method), 134
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian method), 139
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral method), 143
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral method), 147
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_penalties.SpectralPenalties method), 151
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelations method), 157
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape method), 161
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape method), 163
 mprint() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape method), 165
 mprint() (glotaran.model.dataset_descriptor.DatasetDescriptor method), 190
 mprint() (glotaran.model.weight.Weight method), 199

N

name (glotaran.cli.commands.util.ValOrRangeOrList attribute), 171

name (glotaran.cli.main.Cli attribute), 181

nnls () (glotaran.analysis.result.Result property), 53

NON_NEG (glotaran.parameter.parameter.Keys attribute), 201

non_negative () (glotaran.parameter.parameter.Parameter property), 205

non_negative_least_squares () (glotaran.analysis.scheme.Scheme property), 58

normalize () (glotaran.builtin.models.kinetic_image.irf.IrfGaussian property), 78

normalize () (glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian property), 84

normalize () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian property), 139

normalize () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralAnalysis problem.Problem property), 143

normalize () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian property), 147

normalized () (glotaran.builtin.models.kinetic_image.initial_concentration.InitalConcentration problem.Problem method), 74

number_of_data_points () (glotaran.analysis.result.Result property), 53

number_of_function_evaluations () (glotaran.analysis.result.Result property), 53

number_of_jacobian_evaluations () (glotaran.analysis.result.Result property), 54

number_of_variables () (glotaran.analysis.result.Result property), 54

O

OnlyConstraint (class in glotaran.builtin.models.kinetic_spectrum.spectral_constraint), 129

optimization_method () (glotaran.analysis.scheme.Scheme property), 58

optimize () (in module glotaran.analysis.optimize), 31

optimize_cmd () (in module glotaran.cli.commands.optimize), 167

optimize_problem () (in module glotaran.analysis.optimize), 31

optimized_parameters () (glotaran.analysis.result.Result property), 54

P

Parameter (class in glotaran.parameter.parameter), 201

parameter () (glotaran.builtin.models.kinetic_image.irf.IrfGaussian method), 78

parameter () (glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian method), 84

parameter () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian method), 139

parameter () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian method), 143

parameter () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralAnalysis problem.Problem property), 147

parameter () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian method), 151

parameter () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian method), 157

parameter_history () (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralAnalysis problem.Problem property), 45

parameters () (glotaran.analysis.scheme.Scheme property), 58

parameters () (glotaran.builtin.models.kinetic_image.initial_concentration.InitalConcentration problem.Problem property), 74

params (glotaran.cli.main.Cli attribute), 181

parse_args () (glotaran.cli.main.Cli method), 181

parse_spec () (in module glotaran.parse.parser), 216

parse_yaml () (in module glotaran.parse.parser), 216

parse_yaml_file () (in module glotaran.parse.parser), 216

plugin_list_cmd () (in module glotaran.cli.commands.pluginlist), 167

pop () (glotaran.parameter.parameter_group.ParameterGroup method), 214

popitem () (glotaran.parameter.parameter_group.ParameterGroup method), 214

prepare_time_trace_dataset () (in module glotaran.io.prepare_dataset), 182

print_cmd () (in module glotaran.cli.commands.print), 167

Problem (class in glotaran.analysis.problem), 36

problem_list () (glotaran.analysis.scheme.Scheme method), 58

problem_list () (glotaran.builtin.models.kinetic_image.kinetic_image method), 105

problem_list () (glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum method), 126

problem_list () (glotaran.model.base_model.Model method), 188

ProblemDescriptor (class in

glotaran.analysis.problem), 45

R

rates() (*glotaran.builtin.models.kinetic_image.k_matrix.KMatrix* method), 90

read() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile* method), 64

read() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.TimeExplicitFile* method), 66

read() (*glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile* method), 69

read_ascii_time_trace() (in module *glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file*), 61

read_data_file() (in module *glotaran.io.reader*), 183

read_sdt() (in module *glotaran.builtin.file_formats.sdt.sdt_file_reader*), 70

Reader (class in *glotaran.io.reader*), 183

reduced() (*glotaran.builtin.models.kinetic_image.k_matrix.KMatrix* method), 90

reduced_chi_square() (*glotaran.analysis.result.Result* property), 54

reduced_clp_labels() (*glotaran.analysis.problem.Problem* property), 45

reduced_clps() (*glotaran.analysis.problem.Problem* property), 45

reduced_matrices() (*glotaran.analysis.problem.Problem* property), 45

register_model() (in module *glotaran.parse.register*), 217

reset() (*glotaran.analysis.problem.Problem* method), 45

residual_nnls() (in module *glotaran.analysis.nnls*), 30

residual_variable_projection() (in module *glotaran.analysis.variable_projection*), 59

residuals() (*glotaran.analysis.problem.Problem* property), 45

resolve_command() (*glotaran.cli.main.Cli* method), 181

Result (class in *glotaran.analysis.result*), 48

result_callback (*glotaran.cli.main.Cli* attribute), 181

resultcallback() (*glotaran.cli.main.Cli* method), 181

retrieve_clp_function (*glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel* attribute), 106

retrieve_clp_function() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel* method), 126

retrieve_decay_associated_data() (in module *glotaran.builtin.models.kinetic_image.kinetic_image_result.KineticImageResult*), 107

retrieve_explicit_file() (in module *glotaran.builtin.models.kinetic_image.kinetic_image_result*), 107

retrieve_related_clps() (in module *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result.KineticSpectrumResult*), 154

retrieve_species_associated_data() (in module *glotaran.builtin.models.kinetic_image.kinetic_image_result.KineticImageResult*), 107

retrieve_spectral_clps() (in module *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel*), 115

root_group() (*glotaran.parameter.parameter_group.ParameterGroup* property), 214

root_mean_square_error() (*glotaran.analysis.result.Result* property), 54

S

sanitize_dict_keys() (in module *glotaran.parse.util*), 218

sanitize_dict_values() (in module *glotaran.parse.util*), 219

sanitize_list_with_broken_tuples() (in module *glotaran.parse.util*), 219

sanitize_yaml() (in module *glotaran.parse.util*), 219

save() (*glotaran.analysis.result.Result* method), 54

save_parameters_for_history() (*glotaran.analysis.problem.Problem* method), 45

scale() (*glotaran.builtin.models.kinetic_image.irf.IrfGaussian* property), 78

scale() (*glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian* property), 84

scale() (*glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor* property), 93

scale() (*glotaran.builtin.models.kinetic_image.kinetic_image_megacomplex_descriptor.KineticImageMegacomplexDescriptor* property), 97

scale() (*glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor* property), 111

scale() (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussian* property), 139

scale() (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral* property), 143

scale() (*glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectral* property), 147

scale() (*glotaran.model.dataset_descriptor.DatasetDescriptor* property), 190

Scheme (class in `glotaran.analysis.scheme`), 55
 scheme() (`glotaran.analysis.problem.Problem` property), 45
 scheme() (`glotaran.analysis.result.Result` property), 54
 select_data() (in module `glotaran.cli.commands.util`), 168
 select_name() (in module `glotaran.cli.commands.util`), 168
 set_dataset() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 set_dataset() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_explicit_axis() (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile` method), 64
 set_explicit_axis() (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.ExplicitFile` method), 66
 set_explicit_axis() (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile` method), 69
 set_from_group() (`glotaran.parameter.parameter.Parameter` method), 205
 set_from_label_and_value_arrays() (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
 set_initial_concentration() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 set_initial_concentration() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_irf() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 set_irf() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_k_matrix() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 set_k_matrix() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_megacomplex() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 set_megacomplex() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_shape() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 set_value_from_optimization() (`glotaran.parameter.parameter.Parameter` method), 205
 setdefault() (`glotaran.parameter.parameter_group.ParameterGroup` method), 214
 setter() (`glotaran.model.property.ModelProperty` method), 196
 shape() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` property), 111
 shape() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` property), 126
 signature_analysis() (in module `glotaran.cli.commands.util`), 169
 simulate() (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` method), 106
 simulate() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` method), 126
 simulate() (`glotaran.model.base_model.Model` method), 188
 source() (`glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EquationPenalty` method), 151
 source_intervals() (`glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EquationPenalty` method), 151
 spectral_constraints() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` property), 127
 spectral_matrix() (in module `glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel`), 115
 spectral_relations() (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` property), 127
 SpectralConstraint (class in `glotaran.builtin.models.kinetic_spectrum.spectral_constraints`), 131
 SpectralShape (class in `glotaran.builtin.models.kinetic_spectrum.spectral_shape`), 158
 SpectralShape (class in `glotaran.builtin.models.kinetic_spectrum.spectral_shape`), 161
 split_envvar_value() (`glotaran.cli.commands.util.ValOrRangeOrList` method), 171
 standard_error() (`glotaran.parameter.parameter.Parameter` property), 205
 string_to_tuple() (in module `glotaran.parse.util`), 219

success() (glotaran.analysis.result.Result property), 54
 update_parameter_expression() (glotaran.parameter.parameter_group.ParameterGroup method), 215

T

target() (glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty property), 151
 target() (glotaran.builtin.models.kinetic_spectrum.spectral_penalties.SpectralRelation property), 157
 target_intervals() (glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty property), 151
 termination_reason() (glotaran.analysis.result.Result property), 54
 time_explicit (glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.DataType attribute), 62
 TimeExplicitFile (class in glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file), 64
 times() (glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthTimeExplicitFile.kinetic_image.irf.IrfGaussian method), 69
 to_csv() (glotaran.parameter.parameter_group.ParameterGroup method), 214
 to_dataframe() (glotaran.parameter.parameter_group.ParameterGroup method), 215
 transformed_expression() (glotaran.parameter.parameter_group.Parameter property), 205
 type() (glotaran.builtin.models.kinetic_image.irf.IrfGaussian method), 93
 type() (glotaran.builtin.models.kinetic_image.irf.IrfMeasured method), 97
 type() (glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian method), 106
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_constraints.EqualAreaConstraint property), 131
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_constraints.ZeroConstraint property), 134
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussianCoherentArtifact property), 139
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralGaussian method), 143
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian method), 147
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShapeGaussian property), 161
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShapeOne property), 163
 type() (glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShapeZero property), 165

U

update() (glotaran.parameter.parameter_group.ParameterGroup method), 161
 method), 215

`validate()` (`glotaran.builtin.models.kinetic_spectrum.spectral_shape_models.kinetic_spectrum.spectral_irf.IrfSpectralShapeModel` property), 143
`validate()` (`glotaran.builtin.models.kinetic_spectrum.spectral_shape_models.kinetic_spectrum.spectral_irf.IrfSpectralShapeZeroModel` property), 147
`validate()` (`glotaran.model.base_model.Model` property), 161
`validate()` (`glotaran.model.dataset_descriptor.DatasetDescriptor` property), 190
`validate()` (`glotaran.model.property.ModelProperty` property), 139
`validate()` (`glotaran.model.weight.Weight` property), 199
`validate_cmd()` (`in module glotaran.cli.commands.validate`), 172
`ValOrRangeOrList` (`class in glotaran.cli.commands.util`), 169
`value()` (`glotaran.model.weight.Weight` property), 199
`value()` (`glotaran.parameter.parameter.Parameter` property), 205
`values()` (`glotaran.parameter.parameter_group.ParameterGroup` property), 215
`VARY` (`glotaran.parameter.parameter.Keys` attribute), 201
`vary()` (`glotaran.parameter.parameter.Parameter` property), 205

W

`wavelength_explicit` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.DataType` attribute), 62
`WavelengthExplicitFile` (`class in glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile`), 67
`wavelengths()` (`glotaran.builtin.file_formats.ascii.wavelength_time_explicit_file.WavelengthExplicitFile` method), 69
`Weight` (`class in glotaran.model.weight`), 197
`weight` (`glotaran.analysis.problem.GroupedProblem` attribute), 33
`weight` (`glotaran.analysis.problem.ProblemDescriptor` attribute), 47
`weight()` (`glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty` property), 151
`weighted_residuals()` (`glotaran.analysis.problem.Problem` property), 45
`weights()` (`glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel` property), 106
`weights()` (`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.KineticSpectrumModel` property), 127
`width()` (`glotaran.builtin.models.kinetic_image.irf.IrfGaussian` property), 78
`width()` (`glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian` property), 84
`width()` (`glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfGaussianCoherentArtifact` property), 139