pyglotaran Documentation

Release v0.7.1

Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum

2023-07-28

CONTENTS:

CHAPTER

ONE

INTRODUCTION

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

Its features are:

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

1.1 A Note To Glotaran Users

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

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

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

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Python 3.10 or 3.11

2.1.1 Windows

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

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

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

2.2 Stable release

Warning: pyglotaran is early development, so for the moment stable releases are sparse and outdated.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

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

To install pyglotaran, run this command in your terminal:

```
$ pip install pyglotaran
```

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

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

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

2.3 From sources

First you have to install or update some dependencies.

Within a terminal:

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

Alternatively, for Anaconda users:

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

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

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

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

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

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

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

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

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

The following section was generated from `docs/source/notebooks/quickstart/quickstart.ipynb`

CHAPTER

THREE

QUICKSTART/CHEAT-SHEET

To start using `pyglotaran` in your analysis, you only have to import the `Project` class and open a project.

[1]:
```
from glotaran.project import Project

quickstart_project = Project.open("quickstart_project")
quickstart_project
```

[1]:

3.1 Project (*quickstart_project*)

pyglotaran version: `0.7.1`

3.1.1 Data

None

3.1.2 Model

- my_model

3.1.3 Parameters

- my_parameters

3.1.4 Results

None

If the project does not already exist this will create a new project and its folder structure for you. In our case we had only the `models` + `parameters` folders and the `data` + `results` folder were created when opening the project.

[2]:
```
%ls quickstart_project
```

```
data/  models/  parameters/  project.gta  results/
```

Let us get some example data to analyze:

[3]:
```
from glotaran.testing.simulated_data.sequential_spectral_decay import DATASET as my_
↪dataset
```

(continues on next page)

(continued from previous page)

```
my_dataset
```

[3]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
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 -0.01747 -0.006936 ... 2.571 2.31
Attributes:
    source_path:  dataset_1.nc
```

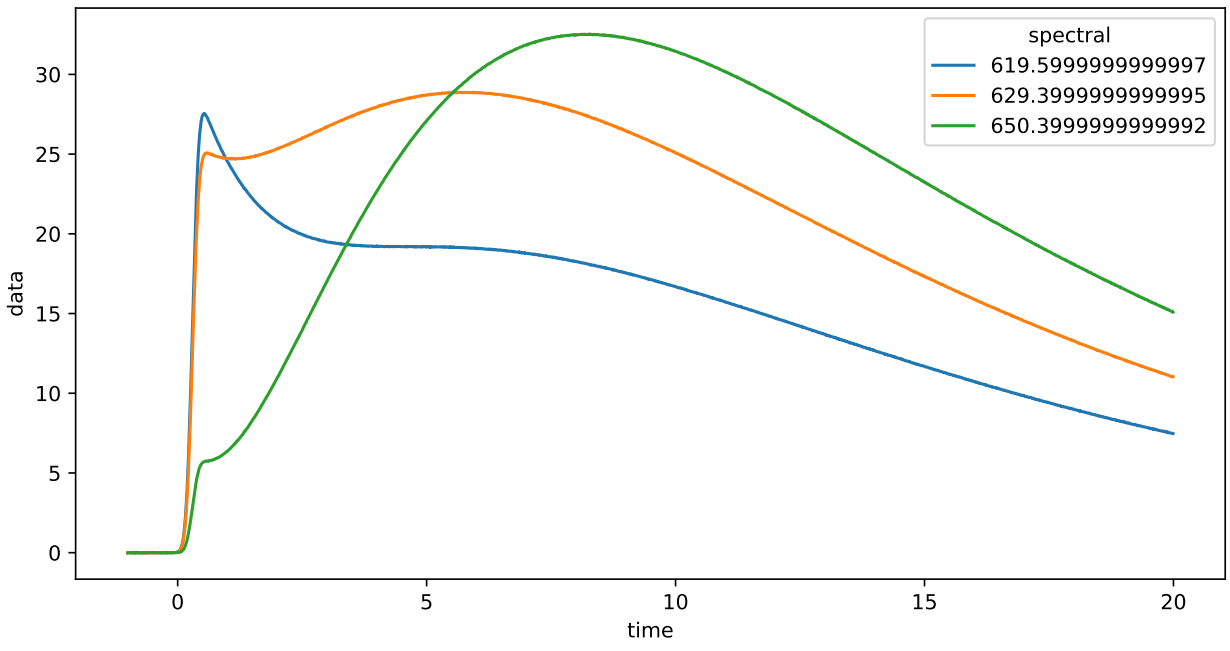
Like all data in `pyglotaran`, the dataset is a xarray.Dataset. You can find more information about the `xarray` library the xarray hompage.

The loaded dataset is a simulated sequential model.

3.2 Plotting raw data

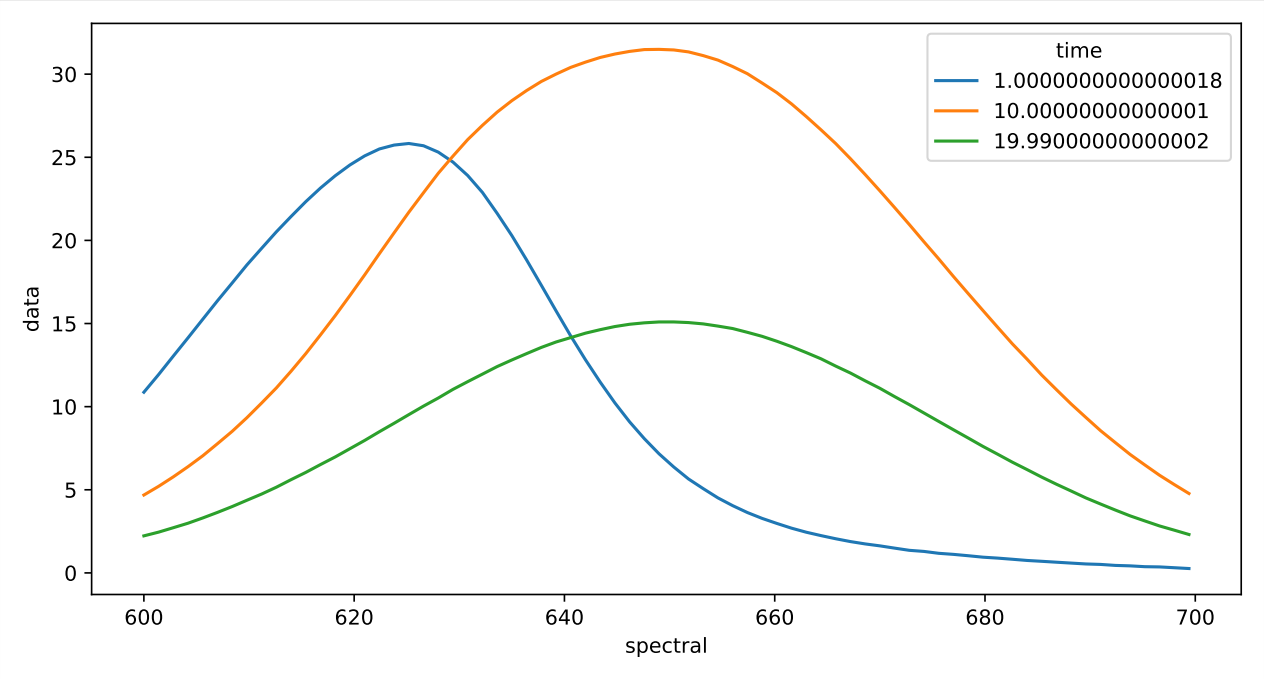
Now lets plot some time traces.

[4]:
```
plot_data = my_dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5);
```



We can also plot spectra at different times.

[5]:
```
plot_data = my_dataset.data.sel(time=[1, 10, 20], method="nearest")
plot_data.plot.line(x="spectral", aspect=2, size=5);
```


time
1.0000000000000018
10.00000000000001
19.99000000000002
data
spectral
0
5
10
15
20
25
30
600
620
640
660
680
700

CHAPTER

FOUR

IMPORT THE DATA INTO YOUR PROJECT

As long as you can read your data into a `xarray.Dataset` or `xarray.DataArray` you can directly import it in to your project.

This will save your data as `NetCDF` (`.nc`) file into the data folder inside of your project with the name that you gave it (here `quickstart_project/data/my_data.nc`).

If the data format you are using is supported by a plugin you can simply copy the file to the data folder of the project (here `quickstart_project/data`).

[6]:
```
quickstart_project.import_data(my_dataset, dataset_name="my_data")
quickstart_project
```

[6]:

4.1 Project (*quickstart_project*)

pyglotaran version: `0.7.1`

4.1.1 Data

- my_data

4.1.2 Model

- my_model

4.1.3 Parameters

- my_parameters

4.1.4 Results

None

After importing our `quickstart_project` is aware of the data that we named `my_data` when importing.

4.2 Preparing data

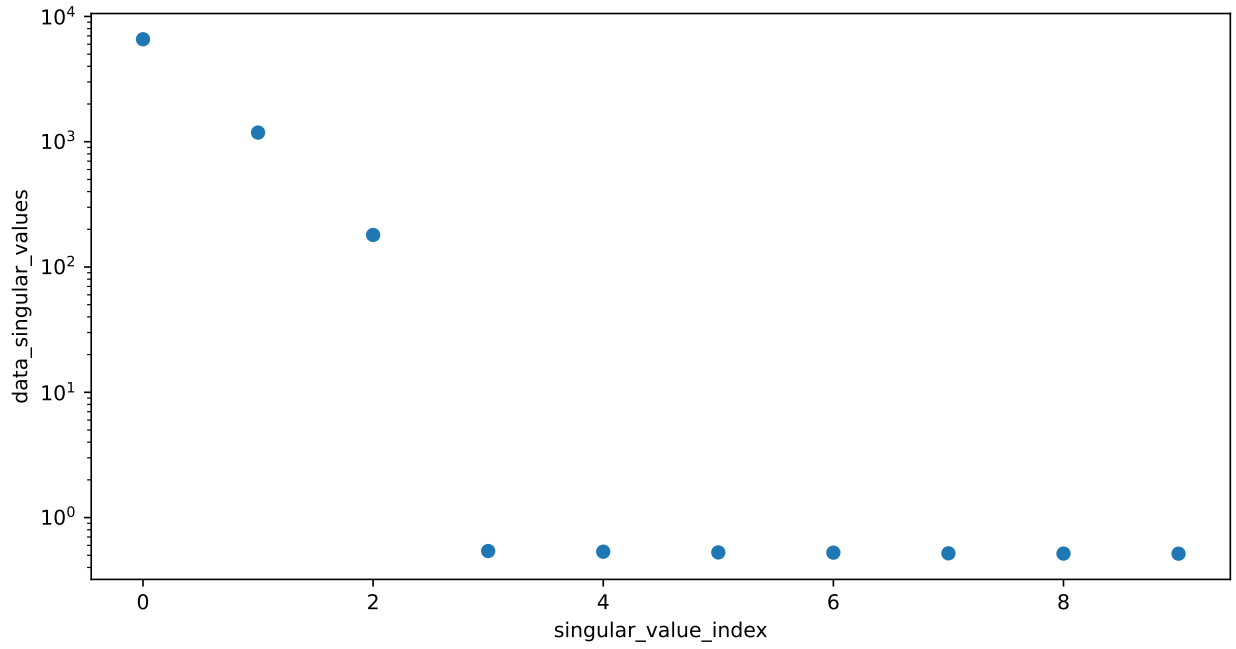
To get an idea about how to model your data, you should inspect the singular value decomposition. As a convenience the `load_data` method has the option to add svd data on the fly.

[7]:
```
dataset_with_svd = quickstart_project.load_data("my_data", add_svd=True)
dataset_with_svd
```

[7]:
```
<xarray.Dataset>
Dimensions:                       (time: 2100, spectral: 72,
                                   left_singular_value_index: 72,
                                   singular_value_index: 72,
                                   right_singular_value_index: 72)
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,
                                singular_value_index, right_singular_value_index
Data variables:
    data                          (time, spectral) float64 -0.01747 ... 2.31
    data_left_singular_vectors    (time, left_singular_value_index) float64 -5...
    data_singular_values          (singular_value_index) float64 6.577e+03 ...
    data_right_singular_vectors   (spectral, right_singular_value_index) float64 ...
Attributes:
    source_path:  /home/docs/checkouts/readthedocs.org/user_builds/pyglotaran...
    loader:       <function load_dataset at 0x7f6cb40a0820>
```

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

[8]:
```
plot_data = dataset_with_svd.data_singular_values.sel(singular_value_index=range(10))
plot_data.plot(yscale="log", marker="o", linewidth=0, aspect=2, size=5);
```



This tells us that our data have at least three components which we need to model.

4.3 Working with models

To analyze our data, we need to create a model.

Create a file called `my_model.yaml` in your projects `models` directory and fill it with the following content.

[9]:
```
quickstart_project.show_model_definition("my_model")
```

[9]:
```
default_megacomplex: decay

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

k_matrix:
  k1:
    matrix:
      (s2, s1): kinetic.1
      (s3, s2): kinetic.2
      (s3, s3): kinetic.3

megacomplex:
  m1:
    k_matrix: [k1]

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

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

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

[10]:
```
quickstart_project.validate("my_model")
```

[10]:
```
Your model is valid.
```

4.4 Working with parameters

Now define some starting parameters. Create a file called `parameters.yaml` in your projects `parameters` directory with the following content.

```
[11]: quickstart_project.show_parameters_definition("my_parameters")
```

```
[11]: input:
        - ["1", 1, { "vary": False }]
        - ["0", 0, { "vary": False }]

      kinetic: [0.51, 0.31, 0.11]

      irf:
        - ["center", 0.31]
        - ["width", 0.11]
```

Note the `{ "vary": False }` which tells `pyglotaran` that those parameters should not be changed.

You can use `validate` method also to check for missing parameters.

```
[12]: quickstart_project.validate("my_model", "my_parameters")
```

```
[12]: 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 load the model and inspect its markdown rendered version.

```
[13]: quickstart_project.load_model("my_model")
```

[13]:

4.4.1 Model

Dataset Groups

- **default**
  - *Label*: default
  - *Residual Function*: variable_projection

K Matrix

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

Megacomplex

- **m1**
  - *Label*: m1

(continues on next page)

(continued from previous page)

  - *Dimension*: time
  - *Type*: decay
  - *K Matrix*: ['k1']

Initial Concentration

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

Irf

- **irf1**
  - *Label*: irf1
  - *Normalize*: True
  - *Backsweep*: False
  - *Type*: gaussian
  - *Center*: irf.center
  - *Width*: irf.width

Dataset

- **my_data**
  - *Label*: my_data
  - *Group*: default
  - *Force Index Dependent*: False
  - *Megacomplex*: ['m1']
  - *Initial Concentration*: input
  - *Irf*: irf1

The same way you should inspect your parameters.

[14]:
```
quickstart_project.load_parameters("my_parameters")
```

[14]:

- **input**:

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	1.000e+00	nan	nan	-inf	inf	False	False	None
0	0.000e+00	nan	nan	-inf	inf	False	False	None

(continues on next page)

(continued from previous page)

- **irf**:

Label	Value	Standard Error	t-value	Minimum	Maximum	Vary	Non-Negative	Expression
center	3.100e-01	nan	nan	-inf	inf	True	False	None
width	1.100e-01	nan	nan	-inf	inf	True	False	None

- **kinetic**:

Label	Value	Standard Error	t-value	Minimum	Maximum	Vary	Non-Negative	Expression
1	5.100e-01	nan	nan	-inf	inf	True	False	None
2	3.100e-01	nan	nan	-inf	inf	True	False	None
3	1.100e-01	nan	nan	-inf	inf	True	False	None

4.5 Optimizing data

Now we have everything together to optimize our parameters.

[15]:
```
result = quickstart_project.optimize("my_model", "my_parameters")
result
```

```
   Iteration     Total nfev        Cost      Cost reduction    Step norm     Optimality
       0              1         1.1176e+04                                    1.73e+06
       1              2         1.5008e+01      1.12e+04       1.96e-02       1.26e+04
       2              3         7.5825e+00      7.43e+00       5.85e-03       1.01e+03
       3              4         7.5786e+00      3.86e-03       1.65e-05       6.29e-02
       4              5         7.5786e+00      1.51e-11       4.58e-09       3.44e-06
Both `ftol` and `xtol` termination conditions are satisfied.
Function evaluations 5, initial cost 1.1176e+04, final cost 7.5786e+00, first-order␣
↪optimality 3.44e-06.
```

[15]:

Optimization Result	
Number of residual evaluation	5
Number of residuals	151200
Number of free parameters	5
Number of conditionally linear parameters	216
Degrees of freedom	150979
Chi Square	1.52e+01
Reduced Chi Square	1.00e-04
Root Mean Square Error (RMSE)	1.00e-02

(continues on next page)

(continued from previous page)

4.5.1 Model

Dataset Groups

- **default**
  - *Label*: default
  - *Residual Function*: variable_projection

K Matrix

- **k1**
  - *Label*: k1
  - *Matrix*: {('s2', 's1'): 'kinetic.1(5.00e-01±6.80e-05, t-value: 7354, initial: 5.10e-01)', ('s3', 's2'): 'kinetic.2(3.00e-01±3.94e-05, t-value: 7609, initial: 3.10e-01)', ('s3', 's3'): 'kinetic.3(1.00e-01±4.23e-06, t-value: 23625, initial: 1.10e-01)'}

Megacomplex

- **m1**
  - *Label*: m1
  - *Dimension*: time
  - *Type*: decay
  - *K Matrix*: ['k1']

Initial Concentration

- **input**
  - *Label*: input
  - *Compartments*: ['s1', 's2', 's3']
  - *Parameters*: ['input.1(1.00e+00, fixed)', 'input.0(0.00e+00, fixed)', 'input.0(0.00e+00, fixed)']
  - *Exclude From Normalize*: []

Irf

- **irf1**
  - *Label*: irf1
  - *Normalize*: True
  - *Backsweep*: False

(continues on next page)

(continued from previous page)

  - *Type*: gaussian
  - *Center*: irf.center(3.00e-01±5.05e-06, t-value: 59448, initial: 3.10e-01)
  - *Width*: irf.width(1.00e-01±6.73e-06, t-value: 14849, initial: 1.10e-01)

Dataset

- **my_data**
  - *Label*: my_data
  - *Group*: default
  - *Force Index Dependent*: False
  - *Megacomplex*: [‘m1’]
  - *Initial Concentration*: input
  - *Irf*: irf1

Each time you run an optimization the result will be saved in the projects results folder.

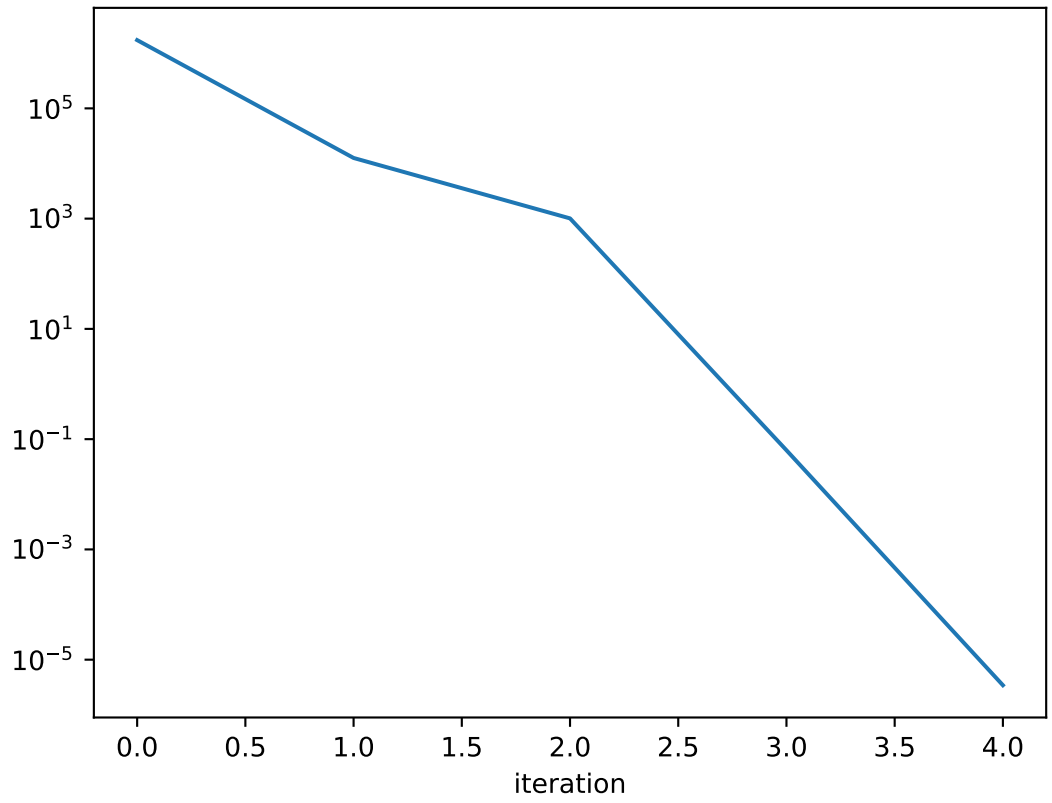
[16]:
```
%ls "quickstart_project/results"
```

```
my_model_run_0000/
```

To visualize how quickly the optimization converged we ca plot the optimality of the `optimization_history`.

[17]:
```
result.optimization_history.data["optimality"].plot(logy=True)
```

[17]:
```
<Axes: xlabel='iteration'>
```



[18]:
```
result.optimized_parameters
```

[18]:

- **input**:

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	1.000e+00	nan	nan	-inf	inf	False	False	None
0	0.000e+00	nan	nan	-inf	inf	False	False	None

- **irf**:

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
center	3.000e-01	5.046e-06	59448	-inf	inf	True	False	None
width	9.999e-02	6.733e-06	14849	-inf	inf	True	False	None

- **kinetic**:

Label	*Value*	*Standard Error*	*t-value*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	5.000e-01	6.799e-05	7354	-inf	inf	True	False	None
2	3.000e-01	3.943e-05	7609	-inf	inf	True	False	None
3	1.000e-01	4.233e-06	23625	-inf	inf	True	False	None

(continues on next page)

(continued from previous page)

You can inspect the data of your `result` by accessing `data` attribute. In our example it only contains our single `my_data` dataset, but it ca contain as many dataset as you analysis needs.

[19]:
```
result.data
```

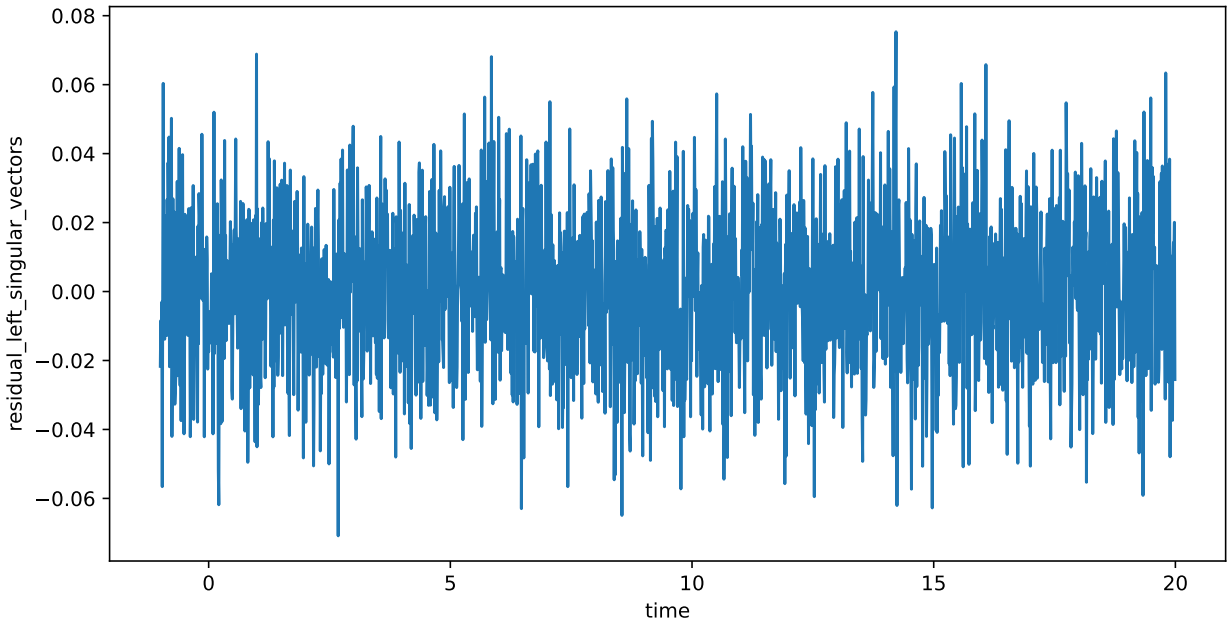
[19]:
```
{'my_data': <xarray.Dataset>}
```

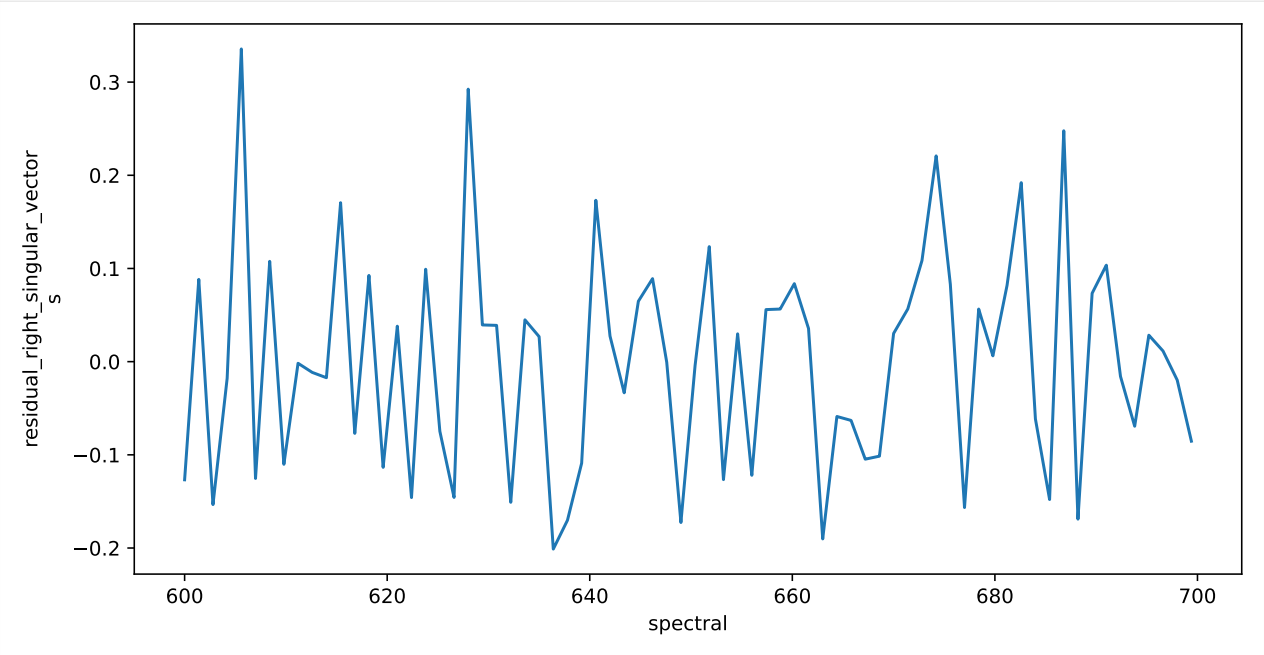
4.6 Visualize the Result

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.

[20]:
```
result_dataset = result.data["my_data"]

residual_left = result_dataset.residual_left_singular_vectors.sel(left_singular_value_
↪index=0)
residual_right = result_dataset.residual_right_singular_vectors.sel(right_singular_value_
↪index=0)
residual_left.plot.line(x="time", aspect=2, size=5)
residual_right.plot.line(x="spectral", aspect=2, size=5);
```




residual_right_singular_vectors
0.3
0.2
0.1
0.0
−0.1
−0.2
600
620
640
660
680
700
spectral

CHAPTER

FIVE

CHANGELOG

5.1 0.7.1 (2023-07-28)

5.1.1 Features

- Python 3.11 support (#1161)

5.1.2 Bug fixes

- Fix coherent artifact clp label duplication (#1292)

5.2 0.7.0 (Unreleased)

5.2.1 BREAKING CHANGE

- Dropped support for Python 3.8 and 3.9 and only support 3.10 (#1135)

5.2.2 Features

- Add optimization history to result and iteration column to parameter history (#1134)
- Complete refactor of model and parameter packages using attrs (#1135)
- Move index dependent calculation to megacomplexes for speed-up (#1175)
- Add PreProcessingPipeline (#1256, #1263)

5.2.3 Minor Improvements:

- Wrap model section in result markdown in details tag for notebooks (#1098)
- Allow more natural column names in pandas parameters file reading (#1174)
- Integrate plugin system into Project (#1229)
- Make yaml the default plugin when passing a folder to save_result and load_result (#1230)
- Allow usage of subfolders in project API for parameters, models and data (#1232)
- Allow import of xarray objects in project API import_data (#1235)

- Add number_of_clps to result and correct degrees_of_freedom calculation (#1249)
- Improve Project API data handling (#1257)
- Deprecate Result.number_of_parameters in favor of Result.number_of_free_parameters (#1262)
- Improve reporting of standard error in case of non_negative constraint in the parameter (#1320)

5.2.4 Bug fixes

- Fix result data overwritten when using multiple dataset_groups (#1147)
- Fix for normalization issue described in #1157 (multi-gaussian irfs and multiple time ranges (streak))
- Fix for crash described in #1183 when doing an optimization using more than 30 datasets (#1184)
- Fix pretty_format_numerical for negative values (#1192)
- Fix yaml result saving with relative paths (#1199)
- Fix model markdown render for items without label (#1213)
- Fix wrong file loading due to partial filename matching in Project (#1212)
- Fix `Project.import_data` path resolving for different script and cwd (#1214)
- Refine project API (#1240)
- Fix search in docs (#1268)

5.2.5 Documentation

- Update quickstart guide to use Project API (#1241)

5.2.6 Deprecations (due in 0.8.0)

- `<model_file>.clp_area_penalties` -> `<model_file>.clp_penalties`
- `glotaran.ParameterGroup` -> `glotaran.Parameters`
- Command Line Interface (removed without replacement) (#1228)
- `Project.generate_model` (removed without replacement)
- `Project.generate_parameters` (removed without replacement)
- `glotaran.project.Result.number_of_data_points` -> `glotaran.project.Result.number_of_residuals`
- `glotaran.project.Result.number_of_parameters` -> `glotaran.project.Result.number_of_free_parameters`

5.2.7 Deprecated functionality removed in this release

- `glotaran.project.Scheme(..., non_negative_least_squares=...)`
- `glotaran.project.Scheme(..., group=...)`
- `glotaran.project.Scheme(..., group_tolerance=...)`
- `<model_file>.non-negative-least-squares: true`
- `<model_file>.non-negative-least-squares: false`
- `glotaran.parameter.ParameterGroup.to_csv(file_name=parameters.csv)`

5.2.8 Maintenance

- Fix wrong comparison in pr_benchmark workflow (#1097)
- Set sourcery-ai target python version to 3.8 (#1095)
- Fix manifest check (#1099)
- Refactor: optimization (#1060)
- Use GITHUB_OUTPUT instead of set-output in github actions (#1166, #1177)
- Add pinned version of odfpy to requirements_dev.txt (#1164)
- Use validation action and validation as a git submodule (#1165)
- Upgrade syntax to py310 using pyupgrade (#1162)
- Remove unused 'type: ignore' (#1168)
- Raise minimum dependency version to releases that support py310 (#1170)
- Make mypy and doc string linters opt out instead of opt in (#1173)

5.3 0.6.0 (2022-06-06)

5.3.1 Features

- Python 3.10 support (#977)
- Add simple decay megacomplexes (#860)
- Feature: Generators (#866)
- Project Class (#869)
- Add clp guidance megacomplex (#1029)

5.3.2 Minor Improvements:

- Add proper repr for DatasetMapping (#957)
- Add SavingOptions to save_result API (#966)
- Add parameter IO support for more formats supported by pandas (#896)
- Apply IRF shift in coherent artifact megacomplex (#992)
- Added IRF shift to result dataset (#994)
- Improve Result, Parameter and ParameterGroup markdown (#1012)
- Add suffix to rate and lifetime and guard for missing datasets (#1022)
- Move simulation to own module (#1041)
- Move optimization to new module glotaran.optimization (#1047)
- Fix missing installation of clp-guide megacomplex as plugin (#1066)
- Add 'extras' and 'full' extras_require installation options (#1089)

5.3.3 Bug fixes

- Fix Crash in optimization_group_calculator_linked when using guidance spectra (#950)
- ParameterGroup.get degrades full_label of nested Parameters with nesting over 2 (#1043)
- Show validation problem if parameters are missing values (default: NaN) (#1076)

5.3.4 Documentation

- Add new logo (#1083, #1087)

5.3.5 Deprecations (due in 0.8.0)

- `glotaran.io.save_result(result, result_path, format_name='legacy')` -> `glotaran.io.save_result(result, Path(result_path) / 'result.yml')`
- `glotaran.analysis.simulation` -> `glotaran.simulation.simulation`
- `glotaran.analysis.optimize` -> `glotaran.optimization.optimize`

5.3.6 Deprecated functionality removed in this release

- `glotaran.ParameterGroup` -> `glotaran.parameter.ParameterGroup`
- `glotaran.read_model_from_yaml` -> `glotaran.io.load_model(..., format_name="yaml_str")`
- `glotaran.read_model_from_yaml_file` -> `glotaran.io.load_model(..., format_name="yaml")`
- `glotaran.read_parameters_from_csv_file` -> `glotaran.io.load_parameters(..., format_name="csv")`
- `glotaran.read_parameters_from_yaml` -> `glotaran.io.load_parameters(..., format_name="yaml_str")`

- `glotaran.read_parameters_from_yaml_file -> glotaran.io.load_parameters(..., format_name="yaml")`
- `glotaran.io.read_data_file -> glotaran.io.load_dataset`
- `result.get_dataset("<dataset_name>") -> result.data["<dataset_name>"]`
- `glotaran.analysis.result -> glotaran.project.result`
- `glotaran.analysis.scheme -> glotaran.project.scheme`

5.3.7 Maintenance

- Improve packaging tooling (#923)
- Exclude test files from duplication checks on sonarcloud (#959)
- Only run check-manifest on the CI (#967)
- Exclude dependabot push CI runs (#978)
- Exclude sourcery AI push CI runs (#1014)
- Auto remove notebook written data when building docs (#1019)
- Change integration tests to use self managed examples action (#1034)
- Exclude pre-commit bot branch from CI runs on push (#1085)

5.4 0.5.1 (2021-12-31)

5.4.1 Bug fixes

- Bugfix Use normalized initial_concentrations in result creation for decay megacomplex (#927)
- Fix save_result crashes on Windows if input data are on a different drive than result (#931)

5.4.2 Maintenance

- Forward port Improve result comparison workflow and v0.4 changelog (#938)
- Forward port of #936 test_result_consistency

5.5 0.5.0 (2021-12-01)

5.5.1 Features

- Feature: Megacomplex Models (#736)
- Feature: Full Models (#747)
- Damped Oscillation Megacomplex (a.k.a. DOAS) (#764)
- Add Dataset Groups (#851)
- Performance improvements (in some cases up to 5x) (#740)

5.5.2 Minor Improvements:

- Add dimensions to megacomplex and dataset_descriptor (#702)
- Improve ordering in k_matrix involved_compartments function (#788)
- Improvements to application of clp_penalties (equal area) (#801)
- Refactor model.from_dict to parse megacomplex_type from dict and add simple_generator for testing (#807)
- Refactor model spec (#836)
- Refactor Result Saving (#841)
- Use ruaml.yaml parser for roundtrip support (#893)
- Refactor Result and Scheme loading/initializing from files (#903)
- Several refactoring in `glotaran.Parameter` (#910)
- Improved Reporting of Parameters (#910, #914, #918)
- Scheme now excepts paths to model, parameter and data file without initializing them first (#912)

5.5.3 Bug fixes

- Fix/cli0.5 (#765)
- Fix compartment ordering randomization due to use of set (#799)
- Fix check_deprecations not showing deprecation warnings (#775)
- Fix and re-enable IRF Dispersion Test (#786)
- Fix coherent artifact crash for index dependent models #808
- False positive model validation fail when combining multiple default megacomplexes (#797)
- Fix ParameterGroup repr when created with 'from_list' (#827)
- Fix for DOAS with reversed oscillations (negative rates) (#839)
- Fix parameter expression parsing (#843)
- Use a context manager when opening a nc dataset (#848)
- Disallow xarray versions breaking plotting in integration tests (#900)
- Fix 'dataset_groups' not shown in model markdown (#906)

5.5.4 Documentation

- Moved API documentation from User to Developer Docs (#776)
- Add docs for the CLI (#784)
- Fix deprecation in model used in quickstart notebook (#834)

5.5.5 Deprecations (due in 0.7.0)

- glotaran.model.Model.model_dimension -> glotaran.project.Scheme.model_dimension
- glotaran.model.Model.global_dimension -> glotaran.project.Scheme.global_dimension
- <model_file>.type.kinetic-spectrum -> <model_file>.default_megacomplex.decay
- <model_file>.type.spectral-model -> <model_file>.default_megacomplex.spectral
- <model_file>.spectral_relations -> <model_file>.clp_relations
- <model_file>.spectral_relations.compartment -> <model_file>.clp_relations.source
- <model_file>.spectral_constraints -> <model_file>.clp_constraints
- <model_file>.spectral_constraints.compartment -> <model_file>.clp_constraints.target
- <model_file>.equal_area_penalties -> <model_file>.clp_area_penalties
- <model_file>.irf.center_dispersion -> <model_file>.irf.center_dispersion_coefficients
- <model_file>.irf.width_dispersion -> <model_file>.irf.width_dispersion_coefficients
- glotaran.project.Scheme(..., non_negative_least_squares=...) -> <model_file>dataset_groups.default.residual_function
- glotaran.project.Scheme(..., group=...) -> <model_file>dataset_groups.default.link_clp
- glotaran.project.Scheme(..., group_tolerance=...) -> glotaran.project.Scheme(..., clp_link_tolerance=...)
- <scheme_file>.maximum-number-function-evaluations -> <scheme_file>.maximum_number_function_evaluations
- <model_file>.non-negative-least-squares: true -> <model_file>dataset_groups.default.residual_function: non_negative_least_squares
- <model_file>.non-negative-least-squares: false -> <model_file>dataset_groups.default.residual_function: variable_projection
- glotaran.parameter.ParameterGroup.to_csv(file_name=parameters.csv) -> glotaran.io.save_parameters(parameters, file_name=parameters.csv)

5.5.6 Maintenance

- Fix Performance Regressions (between version) (#740)
- Add integration test result validation (#754)
- Add more QA tools for parts of glotaran (#739)
- Fix interrogate usage (#781)
- Speedup PR benchmark (#785)
- Use pinned versions of dependencies to run integration CI tests (#892)
- Move megacomplex integration tests from root level to megacomplexes (#894)
- Fix artifact download in pr_benchmark_reaction workflow (#907)

5.6 0.4.2 (2021-12-31)

5.6.1 Bug fixes

- Backport of bugfix #927 discovered in PR #860 related to initial_concentration normalization when saving results (#935).

5.6.2 Maintenance

- Updated 'gold standard' result comparison reference (old -> new)
- Refine test_result_consistency (#936).

5.7 0.4.1 (2021-09-07)

5.7.1 Features

- Integration test result validation (#760)

5.7.2 Bug fixes

- Fix unintended saving of sub-optimal parameters (0ece818, backport from #747)
- Improve ordering in k_matrix involved_compartments function (#791)

5.8 0.4.0 (2021-06-25)

5.8.1 Features

- Add basic spectral model (#672)
- Add Channel/Wavelength dependent shift parameter to irf. (#673)
- Refactored Problem class into GroupedProblem and UngroupedProblem (#681)
- Plugin system was rewritten (#600, #665)
- Deprecation framework (#631)
- Better notebook integration (#689)

5.8.2 Bug fixes

- Fix excessive memory usage in `_create_svd` (#576)
- Fix several issues with KineticImage model (#612)
- Fix exception in sdt reader index calculation (#647)
- Avoid crash in result markdown printing when optimization fails (#630)
- ParameterNotFoundException doesn't prepend '.' if path is empty (#688)
- Ensure Parameter.label is str or None (#678)
- Properly scale StdError of estimated parameters with RMSE (#704)
- More robust covariance_matrix calculation (#706)
- `ParameterGroup.markdown()` independent parametergroups of order (#592)

5.8.3 Plugins

- `ProjectIo` 'folder'/'legacy' plugin to save results (#620)
- `Model` 'spectral-model' (#672)

5.8.4 Documentation

- User documentation is written in notebooks (#568)
- Documentation on how to write a `DataIo` plugin (#600)

5.8.5 Deprecations (due in 0.6.0)

- `glotaran.ParameterGroup` -> `glotaran.parameterParameterGroup`
- `glotaran.read_model_from_yaml` -> `glotaran.io.load_model(..., format_name="yaml_str")`
- `glotaran.read_model_from_yaml_file` -> `glotaran.io.load_model(..., format_name="yaml")`
- `glotaran.read_parameters_from_csv_file` -> `glotaran.io.load_parameters(..., format_name="csv")`
- `glotaran.read_parameters_from_yaml` -> `glotaran.io.load_parameters(..., format_name="yaml_str")`
- `glotaran.read_parameters_from_yaml_file` -> `glotaran.io.load_parameters(..., format_name="yaml")`
- `glotaran.io.read_data_file` -> `glotaran.io.load_dataset`
- `result.save` -> `glotaran.io.save_result(result, ..., format_name="legacy")`
- `result.get_dataset("<dataset_name>")` -> `result.data["<dataset_name>"]`
- `glotaran.analysis.result` -> `glotaran.project.result`
- `glotaran.analysis.scheme` -> `glotaran.project.scheme`
- `model.simulate` -> `glotaran.analysis.simulation.simulate(model, ...)`

5.9 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)

5.10 0.3.2 (2021-02-28)

- Re-release of version 0.3.1 due to packaging issue

5.11 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)

5.12 0.3.0 (2021-02-11)

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

5.13 0.2.0 (2020-12-02)

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

5.14 0.1.0 (2020-07-14)

- Package was renamed to `pyglotaran` on PyPi

5.15 0.0.8 (2018-08-07)

- Changed `nan_policiy` to `omit`

5.16 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartment.

5.17 0.0.6 (2018-08-07)

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

CHAPTER

SIX

AUTHORS

6.1 Development Lead

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

6.2 Contributors

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

6.3 Special Thanks

- Stefan Schuetz
- Sergey P. Laptenok

6.4 Supervision

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

6.5 Original publications

1. Joris J. Snellenburg, Sergey Laptenok, Ralf Seger, Katharine M. Mullen, Ivo H. M. van Stokkum. "Glotaran: A Java-Based Graphical User Interface for the R Package TIMP". Journal of Statistical Software (2012), Volume 49, Number 3, Pages: 1–22. URL https://dx.doi.org/10.18637/jss.v049.i03
2. Katharine M. Mullen, Ivo H. M. van Stokkum. "TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements". Journal of Statistical Software (2007), Volume 18, Number 3, Pages 1-46, ISSN 1548-7660. URL https://dx.doi.org/10.18637/jss.v018.i03
3. Ivo H. M. van Stokkum, Delmar S. Larsen, Rienk van Grondelle, "Global and target analysis of time-resolved spectra". Biochimica et Biophysica Acta (BBA) - Bioenergetics (2004), Volume 1657, Issues 2–3, Pages 82-104, ISSN 0005-2728. URL https://doi.org/10.1016/j.bbabio.2004.04.011

CHAPTER

SEVEN

OVERVIEW

CHAPTER

EIGHT

DATA IO

CHAPTER

NINE

PLOTTING

CHAPTER

TEN

MODELLING

CHAPTER

ELEVEN

PARAMETER

CHAPTER

TWELVE

OPTIMIZING

CHAPTER

THIRTEEN

PLUGINS

To be as flexible as possible `pyglotaran` uses a plugin system to handle new `Models`, `DataIo` and `ProjectIo`. Those plugins can be defined by `pyglotaran` itself, the user or a 3rd party plugin package.

13.1 Builtin plugins

13.1.1 Models

- `KineticSpectrumModel`
- `KineticImageModel`

13.1.2 Data Io

Plugins reading and writing data to and from xarray.Dataset or xarray.DataArray.

- `AsciiDataIo`
- `NetCDFDataIo`
- `SdtDataIo`

13.1.3 Project Io

Plugins reading and writing, `Model`,:class:*Schema*,:class:*ParameterGroup* or `Result`.

- `YmlProjectIo`
- `CsvProjectIo`
- `FolderProjectIo`

13.2 Reproducibility and plugins

With a plugin ecosystem there always is the possibility that multiple plugins try register under the same format/name. This is why plugins are registered at least twice. Once under the name the developer intended and secondly under their full name (full import path). This allows to ensure that a specific plugin is used by manually specifying the plugin, so if someone wants to run your analysis the results will be reproducible even if they have conflicting plugins installed. You can gain all information about the installed plugins by calling the corresponding `*_plugin_table` function with both options (`plugin_names` and `full_names`) set to true. To pin a used plugin use the corresponding `set_*_plugin` function with the intended name (`format_name`/`model_name`) and the full name (`full_plugin_name`) of the plugin to use.

If you wanted to ensure that the pyglotaran builtin plugin is used for `sdt` files you could add the following lines to the beginning of your analysis code.

```
from glotaran.io import set_data_plugin
set_data_plugin("sdt", "glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo_sdt")
```

13.2.1 Models

The functions for model plugins are located in `glotaran.model` and called `model_plugin_table` and `set_model_plugin`.

13.2.2 Data Io

The functions for data io plugins are located in `glotaran.io` and called `data_io_plugin_table` and `set_data_plugin`.

13.2.3 Project Io

The functions for project io plugins are located in `glotaran.io` and called `project_io_plugin_table` and `set_project_plugin`.

13.3 3rd party plugins

Plugins not part of `pyglotaran` itself.

- Not yet, why not be the first? Tell us about your plugin and we will feature it here.

CHAPTER

FOURTEEN

COMMAND-LINE INTERFACE

14.1 glotaran

The glotaran CLI main function.

```
glotaran [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

14.1.1 optimize

Optimizes a model. e.g.: glotaran optimize –

```
glotaran optimize [OPTIONS] [SCHEME_FILE]
```

Options

-dfmt, --dataformat <dataformat>
The input format of the data. Will be inferred from extension if not set.

Options
ascii | nc | sdt

-d, --data <data>
Path to a dataset in the form '–data DATASET_LABEL PATH_TO_DATA'

-o, --out <out>
Path to an output directory.

-ofmt, --outformat <outformat>
The format of the output.

Default
`folder`

Options
folder | legacy | yaml

-n, --nfev <nfev>
Maximum number of function evaluations.

--nnls
Use non-negative least squares.

-y, --yes
Don't ask for confirmation.

-p, --parameters_file <parameters_file>
(optional) Path to parameter file.

-m, --model_file <model_file>
Path to model file.

Arguments

SCHEME_FILE
Optional argument

14.1.2 pluginlist

Prints a list of installed plugins.

```
glotaran pluginlist [OPTIONS]
```

14.1.3 print

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

```
glotaran print [OPTIONS] [SCHEME_FILE]
```

Options

-p, --parameters_file <parameters_file>
(optional) Path to parameter file.

-m, --model_file <model_file>
Path to model file.

Arguments

SCHEME_FILE
Optional argument

14.1.4 validate

Validates a model file and optionally a parameter file.

```
glotaran validate [OPTIONS] [SCHEME_FILE]
```

Options

-p, --parameters_file <parameters_file>
: (optional) Path to parameter file.

-m, --model_file <model_file>
: Path to model file.

Arguments

SCHEME_FILE
: Optional argument

CHAPTER

FIFTEEN

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:

15.1 Types of Contributions

15.1.1 Report Bugs

Report bugs at https://github.com/glotaran/pyglotaran/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

15.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

15.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

15.1.4 Write Documentation

pyglotaran could always use more documentation, whether as part of the official pyglotaran docs, in docstrings, or even on the web in blog posts, articles, and such. If you are writing docstrings please use the NumPyDoc style to write them.

15.1.5 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 :)

15.2 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.

9. Add the change referring the pull request (`(#<PR_nr>)`) to `changelog.md`. If you are in doubt in which section your pull request belongs, just ask a maintainer what they think where it belongs.

Note: By default pull requests will use the template located at `.github/PULL_REQUEST_TEMPLATE.md`. But we also provide custom tailored templates located inside of `.github/PULL_REQUEST_TEMPLATE`. Sadly the GitHub Web Interface doesn't provide an easy way to select them as it does for issue templates (see this comment for more details).

To use them you need to add the following query parameters to the url when creating the pull request and hit enter:

- Feature PR: `?expand=1&template=feature_PR.md`
- Bug Fix PR: `?expand=1&template=bug_fix_PR`
- Documentation PR: `?expand=1&template=docs_PR.md`

15.3 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 3.10 and 3.11 Check your Github Actions `https://github.com/<your_name_here>/pyglotaran/actions` and make sure that the tests pass for all supported Python versions.

15.4 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 are 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.

15.5 Tips

To run a subset of tests:

```
$ py.test tests.test_pyglotaran
```

15.6 Deprecations

Only maintainers are allowed to decide about deprecations, thus you should first open an issue and check back with them if they are ok with deprecating something.

To make deprecations as robust as possible and give users all needed information to adjust their code, we provide helper functions inside the module *glotaran.deprecation*.

The functions you most likely want to use are

- *deprecate()* for functions, methods and classes
- *warn_deprecated()* for call arguments
- *deprecate_module_attribute()* for module attributes
- *deprecate_submodule()* for modules
- *deprecate_dict_entry()* for dict entries
- *raise_deprecation_error()* if the original behavior cannot be maintained

Those functions not only make it easier to deprecate something, but they also check that that deprecations will be removed when they are due and that at least the imports in the warning work. Thus all deprecations need to be tested.

Tests for deprecations should be placed in `glotaran/deprecation/modules/test` which also provides the test helper functions `deprecation_warning_on_call_test_helper` and `changed_import_test_warn`. Since the tests for deprecation are mainly for maintainability and not to test the functionality (those tests should be in the appropriate place) `deprecation_warning_on_call_test_helper` will by default just test that a `GlotaranApiDeprecationWarning` was raised and ignore all raise `Exception` s. An exception to this rule is when adding back removed functionality (which shouldn't happen in the first place but might), which should be implemented in a file under `glotaran/deprecation/modules` and filenames should be like the relative import path from glotaran root, but with _ instead of `.`.

E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

15.6.1 Deprecating a Function, method or class

Deprecating a function, method or class is as easy as adding the `deprecate` decorator to it. Other decorators (e.g. `@staticmethod` or `@classmethod`) should be placed both `deprecate` in order to work.

Listing 1: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

@deprecate(
    deprecated_qual_name_usage="glotaran.some_module.function_to_deprecate(filename)",
```

(continues on next page)

(continued from previous page)

```
    new_qual_name_usage='glotaran.some_module.new_function(filename, format_name="legacy
↪")',
    to_be_removed_in_version="0.6.0",
)
def function_to_deprecate(*args, **kwargs):
    ...
```

15.6.2 Deprecating a call argument

When deprecating a call argument you should use `warn_deprecated` and set the argument to deprecate to a default value (e.g. `"deprecated"`) to check against. Note that for this use case we need to set `check_qual_names=(False, False)` which will deactivate the import testing. This might not always be possible, e.g. if the argument is positional only, so it might make more sense to deprecate the whole callable, just discuss what to do with our trusted maintainers.

Listing 2: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

def function_to_deprecate(args1, new_arg="new_default_behavior", deprecated_arg=
↪"deprecated", **kwargs):
    if deprecated_arg != "deprecated":
        warn_deprecated(
            deprecated_qual_name_usage="deprecated_arg",
            new_qual_name_usage='new_arg="legacy"',
            to_be_removed_in_version="0.6.0",
            check_qual_names=(False, False)
        )
        new_arg = "legacy"
    ...
```

15.6.3 Deprecating a module attribute

Sometimes it might be necessary to remove an attribute (function, class, or constant) from a module to prevent circular imports or just to streamline the API. In those cases you would use `deprecate_module_attribute` inside a module `__getattr__` function definition. This will import the attribute from the new location and return it when an import or use is requested.

Listing 3: glotaran/old_package/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "deprecated_attribute":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.old_package.deprecated_attribute",
            new_qual_name="glotaran.new_package.new_attribute_name",
            to_be_removed_in_version="0.6.0",
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_name}")
```

15.6.4 Deprecating a submodule

For a better logical structure, it might be needed to move modules to a different location in the project. In those cases, you would use `deprecate_submodule`, which imports the module from the new location, add it to `sys.modules` and as an attribute to the parent package.

Listing 4: glotaran/old_package/__init__.py

```
from glotaran.deprecation import deprecate_submodule

module_name = deprecate_submodule(
    deprecated_module_name="glotaran.old_package.module_name",
    new_module_name="glotaran.new_package.new_module_name",
    to_be_removed_in_version="0.6.0",
)
```

15.6.5 Deprecating dict entries

The possible dict deprecation actions are:

- Swapping of keys `{"foo": 1} -> {"bar": 1}` (done via `swap_keys=("foo", "bar")`)
- Replacing of matching values `{"foo": 1} -> {"foo": 2}` (done via `replace_rules=({"foo": 1}, {"foo": 2})`)
- Replacing of matching values and swapping of keys `{"foo": 1} -> {"bar": 2}` (done via `replace_rules=({"foo": 1}, {"bar": 2})`)

For full examples have a look at the examples from the docstring (*deprecate_dict_entry()*).

15.6.6 Deprecation Errors

In some cases deprecations cannot have a replacement with the original behavior maintained. This will be mostly the case when at this point in time and in the object hierarchy there isn't enough information available to calculate the appropriate values. Rather than using a 'dummy' value not to break the API, which could cause undefined behavior down the line, those cases should throw an error which informs the users about the new usage. In general this should only be used if it is unavoidable due to massive refactoring of the internal structure and tried to avoid by any means in a reasonable context.

If you have one of those rare cases you can use *raise_deprecation_error()*.

15.7 Testing Result consistency

To test the consistency of results locally you need to clone the pyglotaran-examples and run them:

```
$ git clone https://github.com/glotaran/pyglotaran-examples
$ cd pyglotaran-examples
$ python scripts/run_examples.py run-all --headless
```

Note: Make sure you got the the latest version (`git pull`) and are on the correct branch for both `pyglotaran` and `pyglotaran-examples`.

The results from the examples will be saved in you home folder under `pyglotaran_examples_results`. Those results than will be compared to the 'gold standard' defined by the maintainers.

To test the result consistency run:

```
$ pytest validation/pyglotaran-examples/test_result_consistency.py
```

If needed this will clone the 'gold standard' results to the folder `comparison-results`, update them and test your current results against them.

15.8 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in `changelog.md`), 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. `v0.3.0` since it might be included in a citation.

Github Actions will then deploy to PyPI if the tests pass.

CHAPTER

SIXTEEN

API DOCUMENTATION

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

glotaran	Glotaran package root.

16.1 glotaran

Glotaran package root.

Modules

glotaran.analysis	This package contains functions for model simulation and fitting.
glotaran.builtin	This package contains builtin plugins.
glotaran.cli	
glotaran.deprecation	Deprecation helpers and place to put deprecated implementations till removing.
glotaran.io	Functions for data IO.
glotaran.model	The glotaran model package.
glotaran.optimization	This package contains functions for optimization.
glotaran.parameter	The glotaran parameter package.
glotaran.plugin_system	Plugin system package containing all plugin related implementations.
glotaran.project	The glotaran project package.
glotaran.simulation	Package containing code for simulation of dataset models.
glotaran.testing	Testing framework package for glotaran itself and plugins.
glotaran.typing	Glotaran specific typing module.
glotaran.utils	Glotaran utility function/class package.

16.1.1 analysis

This package contains functions for model simulation and fitting.

16.1.2 builtin

This package contains builtin plugins.

Modules

glotaran.builtin.io	Package containing the builtin IO plugins.
glotaran.builtin.megacomplexes	

io

Package containing the builtin IO plugins.

Modules

glotaran.builtin.io.ascii	
glotaran.builtin.io.folder	Plugin to dump pyglotaran object as files in a folder.
glotaran.builtin.io.netCDF	Package containing the NetCDF4 Data IO plugin.
glotaran.builtin.io.pandas	Pandas io package.
glotaran.builtin.io.sdt	Package containing the SDT Data IO plugin.
glotaran.builtin.io.yml	Package containing the YAML Data and Project IO plugins.

ascii

Modules

glotaran.builtin.io.ascii.wavelength_time_explicit_file	

wavelength_time_explicit_file

Functions

Summary

get_data_file_format	
get_interval_number	

get_data_file_format

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_data_file_format**(*line*)

get_interval_number

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_interval_number**(*line*)

Classes

Summary

AsciiDataIo	Initialize a Data IO plugin with the name of the format.
DataFileType	An enumeration.
ExplicitFile	Abstract class representing either a time- or wavelength-explicit file.
TimeExplicitFile	Represents a time explicit file
WavelengthExplicitFile	Represents a wavelength explicit file

AsciiDataIo

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**AsciiDataIo**(*format_name: str*)

Bases: *DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Reads an ascii file in wavelength- or time-explicit format.
save_dataset	Save data from xarray.Dataset to a file.

load_dataset

AsciiDataIo.**load_dataset**(*file_name: str*, *, *prepare: bool = True*) → Dataset | DataArray

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

save_dataset

AsciiDataIo.**save_dataset**(*dataset: DataArray | Dataset*, *file_name: str*, *, *comment: str = ''*, *file_format: DataFileType = DataFileType.time_explicit*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED

Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*, *, *prepare: bool = True*) → Dataset | DataArray

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

save_dataset(*dataset: DataArray | Dataset*, *file_name: str*, *, *comment: str = ''*, *file_format: DataFileType = DataFileType.time_explicit*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

DataFileType

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**DataFileType**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

time_explicit
wavelength_explicit

time_explicit

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

wavelength_explicit

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

time_explicit = 'Time explicit'

wavelength_explicit = 'Wavelength explicit'

ExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**ExplicitFile**(*filepath: str | None = None*, *dataset: DataArray | None = None*)

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*) → Dataset | 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*, *number_format='%.10e'*)

Methods Documentation

dataset(*prepare: bool = True*) → Dataset | 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*, *number_format='%.10e'*)

TimeExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**TimeExplicitFile**(*filepath: str | None = None*, *dataset: DataArray | None = None*)

Bases: *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*) → Dataset | DataArray

get_data_row

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

get_explicit_axis

TimeExplicitFile.**get_explicit_axis**()

get_format_name

TimeExplicitFile.**get_format_name**()

get_observations

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

get_secondary_axis

TimeExplicitFile.**get_secondary_axis**()

read

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

set_explicit_axis

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

write

TimeExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

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

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axies*)

write(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

WavelengthExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**WavelengthExplicitFile**(*filepath: str | None = None*, *dataset: DataArray | None = None*)

Bases: *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*) → Dataset | 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,
number_format='%.10e')

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → Dataset | 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*, *number_format='%.10e'*)

folder

Plugin to dump pyglotaran object as files in a folder.

Modules

glotaran.builtin.io.folder.folder_plugin	Implementation of the folder Io plugin.

folder_plugin

Implementation of the folder Io plugin.

The current implementation is an exact copy of how `Result.save(path)` worked in glotaran 0.3.x and meant as an compatibility function.

Classes

Summary

FolderProjectIo	Project Io plugin to save result data to a folder.
LegacyProjectIo	Project Io plugin to save result data in a backward compatible manner.

FolderProjectIo

class glotaran.builtin.io.folder.folder_plugin.**FolderProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Project Io plugin to save result data to a folder.

There won't be a serialization of the Result object, but simply a markdown summary output and the important data saved to files.

Initialize a Project IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a Parameters instance from the specs defined in a file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a Parameters instance to a spec file.
save_result	Save the result to a given folder.
save_scheme	Save a Scheme instance to a spec file.

load_model

FolderProjectIo.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED
Parameters
file_name (*str*) – File containing the model specs.
Returns
Model instance created from the file.
Return type
Model

load_parameters

FolderProjectIo.**load_parameters**(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

Parameters instance created from the file.

Return type

Parameters

load_result

FolderProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme

FolderProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

FolderProjectIo.**save_model**(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

FolderProjectIo.**save_parameters**(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters

- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

FolderProjectIo.**save_result**(*result:* Result, *result_path: str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*, *used_inside_of_plugin: bool = False*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Note: As a side effect it populates the file path properties of `result` which can be used in other plugins (e.g. the `yml` save_result).

Parameters

- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.
- **used_inside_of_plugin** (*bool*) – Denote that this plugin is used from inside another plugin, if false a user warning will be thrown. , by default False

Returns

List of file paths which were created.

Return type

list[str]

Raises

ValueError – If `result_path` is a file.

save_scheme

FolderProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the model specs.

Returns
Model instance created from the file.

Return type
Model

load_parameters(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
Parameters instance created from the file.

Return type
Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
result_path (*str*) – Path containing the result data.

Returns
Result instance created from the file.

Return type
Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters
- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters
- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*, ***, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*, *used_inside_of_plugin: bool = False*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Note: As a side effect it populates the file path properties of `result` which can be used in other plugins (e.g. the `yml` save_result).

Parameters
- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.
- **used_inside_of_plugin** (*bool*) – Denote that this plugin is used from inside another plugin, if false a user warning will be thrown. , by default False

Returns
List of file paths which were created.

Return type
list[str]

Raises
ValueError – If `result_path` is a file.

save_scheme(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

LegacyProjectIo

class glotaran.builtin.io.folder.folder_plugin.**LegacyProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Project Io plugin to save result data in a backward compatible manner.

Initialize a Project IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a Parameters instance from the specs defined in a file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a Parameters instance to a spec file.
save_result	Save the result to a given folder.
save_scheme	Save a Scheme instance to a spec file.

load_model

LegacyProjectIo.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the model specs.

Returns

Model instance created from the file.

Return type

Model

load_parameters

LegacyProjectIo.**load_parameters**(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

Parameters instance created from the file.

Return type

Parameters

load_result

LegacyProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme

LegacyProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

LegacyProjectIo.**save_model**(*model: Model*, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

LegacyProjectIo.**save_parameters**(*parameters: Parameters*, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters

- **parameters** (*Parameters*) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

LegacyProjectIo.**save_result**(*result:* Result, *result_path: str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to a given folder.

Warning: Deprecated use `glotaran.io.save_result(result, Path(result_path) / 'result.yml')` instead.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `result.yml`: Yaml spec file of the result * `model.yml`: Model spec file. * `scheme.yml`: Scheme spec file. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Parameters
- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.

Returns
List of file paths which were created.

Return type
list[str]

save_scheme

LegacyProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the model specs.

Returns
Model instance created from the file.

Return type
Model

load_parameters(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

Parameters instance created from the file.

Return type

Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters

- **parameters** (*Parameters*) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result:* Result, *result_path: str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to a given folder.

Warning: Deprecated use `glotaran.io.save_result(result, Path(result_path) / 'result.yml')` instead.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `result.yml`: Yaml spec file of the result * `model.yml`: Model spec file. * `scheme.yml`: Scheme spec file. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Parameters
- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.

Returns
List of file paths which were created.

Return type
list[str]

save_scheme(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

netCDF

Package containing the NetCDF4 Data IO plugin.

Modules

glotaran.builtin.io.netCDF.netCDF	Module containing the NetCDF4 Data IO plugin.

netCDF

Module containing the NetCDF4 Data IO plugin.

Classes

Summary

NetCDFDataIo	Plugin for NetCDF4 data io.

NetCDFDataIo

class glotaran.builtin.io.netCDF.netCDF.**NetCDFDataIo**(*format_name: str*)

Bases: *DataIoInterface*

Plugin for NetCDF4 data io.

Initialize a Data IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Load a `*.nc` file into a xarray.Dataset or xarray.DataArray.
save_dataset	Write a xarray.Dataset to the `*.nc` at path `file_name`.

load_dataset

NetCDFDataIo.**load_dataset**(*file_name: str*) → Dataset | DataArray

Load a `*.nc` file into a xarray.Dataset or xarray.DataArray.

Parameters
file_name (*str*) – Path to the `*.nc` file that should be loaded.

Return type
xr.Dataset | xr.DataArray

save_dataset

NetCDFDataIo.**save_dataset**(*dataset: Dataset*, *file_name: str*, *, *data_filters: list[str] | None = None*)

Write a xarray.Dataset to the `*.nc` at path `file_name`.

Parameters
- **dataset** (*xr.Dataset*) – xarray.Dataset that should be written to file.
- **file_name** (*str*) – Path of the file to write `dataset` to.
- **data_filters** (*list[str] | None*) – List of data variable names that should be written to file. Defaults to None.

Methods Documentation

load_dataset(*file_name: str*) → Dataset | DataArray

Load a `*.nc` file into a xarray.Dataset or xarray.DataArray.

Parameters
file_name (*str*) – Path to the `*.nc` file that should be loaded.

Return type
xr.Dataset | xr.DataArray

save_dataset(*dataset: Dataset*, *file_name: str*, ***, *data_filters: list[str] | None = None*)

Write a xarray.Dataset to the `*.nc` at path `file_name`.

Parameters

- **dataset** (*xr.Dataset*) – xarray.Dataset that should be written to file.
- **file_name** (*str*) – Path of the file to write `dataset` to.
- **data_filters** (*list[str] | None*) – List of data variable names that should be written to file. Defaults to None.

pandas

Pandas io package.

Modules

glotaran.builtin.io.pandas.csv	Module containing CSV io support.
glotaran.builtin.io.pandas.tsv	Module containing TSV io support.
glotaran.builtin.io.pandas.xlsx	Module containing Excel like io support.

csv

Module containing CSV io support.

Classes

Summary

CsvProjectIo	Plugin for CSV data io.

CsvProjectIo

class glotaran.builtin.io.pandas.csv.**CsvProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Plugin for CSV data io.

Initialize a Project IO plugin with the name of the format.

Parameters

format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Load parameters from CSV file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a `Parameters` to a CSV file.
save_result	Save a Result instance to a spec file.
save_scheme	Save a Scheme instance to a spec file.

load_model

CsvProjectIo.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the model specs.

Returns

Model instance created from the file.

Return type

Model

load_parameters

CsvProjectIo.**load_parameters**(*file_name: str*, *sep: str = ','*) → *Parameters*

Load parameters from CSV file.

Parameters

- **file_name** (*str*) – Name of file to be loaded.
- **sep** (*str*) – Other separators can be used optionally., by default ‘,’

Return type

class:`Parameters

load_result

CsvProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme

CsvProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

CsvProjectIo.**save_model**(*model: Model*, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

CsvProjectIo.**save_parameters**(*parameters: Parameters*, *file_name: str*, ***, *sep: str = ','*, *as_optimized: bool = True*, *replace_infinfinity: bool = True*) → None

Save a Parameters to a CSV file.

Parameters

- **parameters** (Parameters) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.
- **sep** (*str*) – Other separators can be used optionally., by default ‘,’
- **as_optimized** (*bool*) – Weather to include properties which are the result of optimization.
- **replace_infinfinity** (*bool*) – Weather to replace infinity values with empty strings.

save_result

CsvProjectIo.**save_result**(*result: Result*, *result_path: str*, ***, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme

CsvProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the model specs.

Returns

Model instance created from the file.

Return type

Model

load_parameters(*file_name: str*, *sep: str = ','*) → *Parameters*

Load parameters from CSV file.

Parameters

- **file_name** (*str*) – Name of file to be loaded.
- **sep** (*str*) – Other separators can be used optionally., by default ‘,’

Return type

class:`Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name:* *str*, *, *sep:* *str* = ',', *as_optimized:* *bool* = *True*, *replace_infinfinity:* *bool* = *True*) → None

Save a Parameters to a CSV file.

Parameters

- **parameters** (Parameters) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.
- **sep** (*str*) – Other separators can be used optionally., by default ‘,’
- **as_optimized** (*bool*) – Weather to include properties which are the result of optimization.
- **replace_infinfinity** (*bool*) – Weather to replace infinity values with empty strings.

save_result(*result:* Result, *result_path:* *str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme(*scheme:* Scheme, *file_name:* *str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

tsv

Module containing TSV io support.

Classes

Summary

TsvProjectIo	Plugin for TSV data io.

TsvProjectIo

class glotaran.builtin.io.pandas.tsv.**TsvProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Plugin for TSV data io.

Initialize a Project IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Load parameters from TSV file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a Parameters to a TSV file.
save_result	Save a Result instance to a spec file.
save_scheme	Save a Scheme instance to a spec file.

load_model

TsvProjectIo.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED
Parameters
file_name (*str*) – File containing the model specs.
Returns
Model instance created from the file.
Return type
Model

load_parameters

TsvProjectIo.**load_parameters**(*file_name: str*) → *Parameters*

Load parameters from TSV file.
Parameters
file_name (*str*) – Name of file to be loaded.
Return type
Parameters

load_result

TsvProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme

TsvProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

TsvProjectIo.**save_model**(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

TsvProjectIo.**save_parameters**(*parameters:* Parameters, *file_name: str*, *, *replace_infinfinity: bool = True*) → None

Save a Parameters to a TSV file.

Parameters

- **parameters** (Parameters) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.
- **replace_infinfinity** (*bool*) – Weather to replace infinity values with empty strings.

save_result

TsvProjectIo.**save_result**(*result:* Result, *result_path: str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme

TsvProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the model specs.

Returns

Model instance created from the file.

Return type

Model

load_parameters(*file_name: str*) → *Parameters*

Load parameters from TSV file.

Parameters

file_name (*str*) – Name of file to be loaded.

Return type

Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters: Parameters*, *file_name: str*, *, *replace_infinfinity: bool = True*) → None

Save a `Parameters` to a TSV file.

Parameters

- **parameters** (*Parameters*) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.
- **replace_infinfinity** (*bool*) – Weather to replace infinity values with empty strings.

save_result(*result: Result*, *result_path: str*, *, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (*Result*) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (*SavingOptions*) – Options for the saved result.

save_scheme(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (*Scheme*) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

xlsx

Module containing Excel like io support.

Classes

Summary

ExcelProjectIo	Plugin for Excel like data io.

ExcelProjectIo

class glotaran.builtin.io.pandas.xlsx.**ExcelProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Plugin for Excel like data io.

Initialize a Project IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Load parameters from XLSX file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a `Parameters` to a Excel file.
save_result	Save a Result instance to a spec file.
save_scheme	Save a Scheme instance to a spec file.

load_model

ExcelProjectIo.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED
Parameters
file_name (*str*) – File containing the model specs.
Returns
Model instance created from the file.
Return type
Model

load_parameters

ExcelProjectIo.**load_parameters**(*file_name: str*) → *Parameters*

Load parameters from XLSX file.

Parameters
file_name (*str*) – Name of file to be loaded.

Return type
Parameters

load_result

ExcelProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
result_path (*str*) – Path containing the result data.

Returns
Result instance created from the file.

Return type
Result

load_scheme

ExcelProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

ExcelProjectIo.**save_model**(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters
- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

ExcelProjectIo.**save_parameters**(*parameters:* Parameters, *file_name: str*)

Save a Parameters to a Excel file.

Parameters

- **parameters** (Parameters) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.

save_result

ExcelProjectIo.**save_result**(*result:* Result, *result_path: str*, **, saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme

ExcelProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the model specs.

Returns

Model instance created from the file.

Return type

Model

load_parameters(*file_name: str*) → *Parameters*

Load parameters from XLSX file.

Parameters

file_name (*str*) – Name of file to be loaded.

Return type

Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

result_path (*str*) – Path containing the result data.

Returns

Result instance created from the file.

Return type

Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name: str*)

Save a Parameters to a Excel file.

Parameters

- **parameters** (Parameters) – Parameters to be saved to file.
- **file_name** (*str*) – File to write the parameters to.

save_result(*result:* Result, *result_path: str*, ***, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

sdt

Package containing the SDT Data IO plugin.

Modules

glotaran.builtin.io.sdt.sdt_file_reader	Module containing the SDT Data IO plugin.

sdt_file_reader

Module containing the SDT Data IO plugin.

Classes

Summary

SdtDataIo	Plugin for SDT data io.

SdtDataIo

class glotaran.builtin.io.sdt.sdt_file_reader.**SdtDataIo**(*format_name: str*)

Bases: *DataIoInterface*

Plugin for SDT data io.

Initialize a Data IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Read a **.sdt* file and returns a xarray.Dataset containing its data.
save_dataset	Save data from xarray.Dataset to a file.

load_dataset

SdtDataIo.**load_dataset**(*file_name: str*, *, *index: ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → Dataset

Read a **.sdt* file and returns a xarray.Dataset containing its data.

Parameters

- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*np.ndarray | None*) – 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** (*bool*) – Set true if reading a result from a FLIM measurement. Defaults to False.
- **dataset_index** (*int*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one. Defaults to 0.
- **swap_axis** (*bool*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset. Defaults to False.
- **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.

Return type

xr.Dataset

Raises

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

save_dataset

SdtDataIo.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*, *, *index: ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → Dataset

Read a **.sdt* file and returns a xarray.Dataset containing its data.

Parameters

- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*np.ndarray | None*) – 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** (*bool*) – Set true if reading a result from a FLIM measurement. Defaults to False.
- **dataset_index** (*int*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one. Defaults to 0.
- **swap_axis** (*bool*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset. Defaults to False.

- **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.

Return type
xr.Dataset

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

save_dataset(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED

Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

yml

Package containing the YAML Data and Project IO plugins.

Modules

glotaran.builtin.io.yml.utils	Utility module for `glotaran.builtin.io.yml.yml`.
glotaran.builtin.io.yml.yml	Module containing the YAML Data and Project IO plugins.

utils

Utility module for `glotaran.builtin.io.yml.yml`.

Functions

Summary

load_dict	Load `yaml` code from a file or string and returns the dict interpretation.
write_dict	Write a mapping (e.g.

load_dict

glotaran.builtin.io.yml.utils.**load_dict**(*source: str | Path*, *is_file: bool*) → dict[str, Any]

Load `yaml` code from a file or string and returns the dict interpretation.

Parameters

- **source** (*str | Path*) – Path to a file or string containing the `yaml` code.
- **is_file** (*bool*) – Whether or not `source` is a file.

Return type
dict[str, Any]

write_dict

glotaran.builtin.io.yml.utils.**write_dict**(*data: Mapping[str, Any] | Sequence[Any]*, *file_name: str | Path | None = None*, *offset: int = 0*) → str | None

Write a mapping (e.g. `dict`) or sequence (e.g. `list`) as `yaml` to file or str.

Parameters

- **data** (*Mapping[str, Any] | Sequence[Any]*) – Data that should be converted to `yaml`.
- **file_name** (*str | Path | None*) – Path of the file to write the `yaml` code to. Defaults to None which makes this function return a string.
- **offset** (*int*) – Block indentation level. Defaults to 0 See https://yaml.readthedocs.io/en/latest/detail.html#indentation-of-block-sequences

Returns
String if `file_name` is `None` or `None` if `file_name` is a valid path.

Return type
str | None

yml

Module containing the YAML Data and Project IO plugins.

Classes

Summary

YmlProjectIo	Plugin for YAML project io.

YmlProjectIo

class glotaran.builtin.io.yml.yml.**YmlProjectIo**(*format_name: str*)

Bases: *ProjectIoInterface*

Plugin for YAML project io.

Initialize a Project IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Load a `Model` from a model specification in a yaml file.
load_parameters	Load `Parameters` instance from the specification defined in `file_name`.
load_result	Create a `Result` instance from the specs defined in a file.
load_scheme	Load `Scheme` instance from the specification defined in `file_name`.
save_model	Save a `Model` instance to a specification file.
save_parameters	Save a Parameters instance to a spec file.
save_result	Write a `Result` instance to a specification file and data files.
save_scheme	Write a `Scheme` instance to a specification file `file_name`.

load_model

YmlProjectIo.**load_model**(*file_name: str*) → *Model*

Load a `Model` from a model specification in a yaml file.

Parameters
file_name (*str*) – Path to the model file to read.

Raises
- **ValueError** – If `megacomplex` was not provided in the model specification.
- **ValueError** – If `default_megacomplex` was not provided and any megacomplex is missing the type attribute.

Return type
Model

load_parameters

YmlProjectIo.**load_parameters**(*file_name: str*) → *Parameters*

Load Parameters instance from the specification defined in file_name.

Parameters
file_name (*str*) – File containing the parameter specification.

Return type
Parameters

load_result

YmlProjectIo.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

Parameters
result_path (*str*) – Path containing the result data.

Returns
Result instance created from the saved format.

Return type
Result

load_scheme

YmlProjectIo.**load_scheme**(*file_name: str*) → *Scheme*

Load Scheme instance from the specification defined in file_name.

Parameters
file_name (*str*) – File containing the scheme specification.

Return type
Scheme

save_model

YmlProjectIo.**save_model**(*model:* Model, *file_name: str*)

Save a Model instance to a specification file.

Parameters
- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

YmlProjectIo.**save_parameters**(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters
- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

YmlProjectIo.**save_result**(*result:* Result, *result_path: str*, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Write a `Result` instance to a specification file and data files.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `result.yml`: Yaml spec file of the result * `model.yml`: Model spec file. * `scheme.yml`: Scheme spec file. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `optimization_history.csv`: Parsed table printed by the SciPy optimizer * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Parameters
- **result** (Result) – `Result` instance to write.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for saving the the result.

Returns
List of file paths which were created.

Return type
list[str]

save_scheme

YmlProjectIo.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Write a `Scheme` instance to a specification file `file_name`.

Parameters
- **scheme** (Scheme) – `Scheme` instance to save to file.
- **file_name** (*str*) – Path to the file to write the scheme specification to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Load a `Model` from a model specification in a yaml file.

Parameters
file_name (*str*) – Path to the model file to read.

Raises
- **ValueError** – If `megacomplex` was not provided in the model specification.
- **ValueError** – If `default_megacomplex` was not provided and any megacomplex is missing the type attribute.

Return type
Model

load_parameters(*file_name: str*) → *Parameters*

Load `Parameters` instance from the specification defined in `file_name`.

Parameters
file_name (*str*) – File containing the parameter specification.

Return type
Parameters

load_result(*result_path: str*) → *Result*

Create a `Result` instance from the specs defined in a file.

Parameters
result_path (*str*) – Path containing the result data.

Returns
`Result` instance created from the saved format.

Return type
Result

load_scheme(*file_name: str*) → *Scheme*

Load `Scheme` instance from the specification defined in `file_name`.

Parameters
file_name (*str*) – File containing the scheme specification.

Return type
Scheme

save_model(*model:* Model, *file_name: str*)

Save a `Model` instance to a specification file.

Parameters
- **model** (`Model`) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters
- **parameters** (`Parameters`) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result:* Result, *result_path: str*, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Write a `Result` instance to a specification file and data files.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * `result.md`: The result with the model formatted as markdown text. * `result.yml`: Yaml spec file of the result * `model.yml`: Model spec file. * `scheme.yml`: Scheme spec file. * `initial_parameters.csv`: Initially used parameters. * `optimized_parameters.csv`: The optimized parameter as csv file. * `parameter_history.csv`: Parameter changes over the optimization * `optimization_history.csv`: Parsed table printed by the SciPy optimizer * `{dataset_label}.nc`: The result data for each dataset as NetCDF file.

Parameters
- **result** (`Result`) – `Result` instance to write.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (`SavingOptions`) – Options for saving the the result.

Returns
List of file paths which were created.

Return type
list[str]

save_scheme(*scheme:* Scheme, *file_name: str*)

Write a `Scheme` instance to a specification file `file_name`.

Parameters
- **scheme** (`Scheme`) – `Scheme` instance to save to file.
- **file_name** (*str*) – Path to the file to write the scheme specification to.

megacomplexes

Modules

glotaran.builtin.megacomplexes.baseline
glotaran.builtin.megacomplexes.clp_guide
glotaran.builtin.megacomplexes.coherent_artifact
glotaran.builtin.megacomplexes.damped_oscillation
glotaran.builtin.megacomplexes.decay
glotaran.builtin.megacomplexes.spectral

baseline

Modules

glotaran.builtin.megacomplexes.baseline.baseline_megacomplex

baseline_megacomplex

Classes

Summary

BaselineMegacomplex	Method generated by attrs for class BaselineMegacomplex.

BaselineMegacomplex

class glotaran.builtin.megacomplexes.baseline.baseline_megacomplex.**BaselineMegacomplex**(*, *label: str*, *dimension: str | None = None*, *type: str = 'baseline'*)

Bases: `Megacomplex`

Method generated by attrs for class BaselineMegacomplex.

Attributes Summary

type
dimension
label

type

BaselineMegacomplex.**type: str**

dimension

BaselineMegacomplex.**dimension: str | None**

label

BaselineMegacomplex.**label: str**

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_dataset_model_type	Get the dataset model type.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate_matrix

BaselineMegacomplex.**calculate_matrix**(*dataset_model:* DatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

BaselineMegacomplex.**finalize_data**(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_dataset_model_type

classmethod BaselineMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_item_type

classmethod BaselineMegacomplex.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod BaselineMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod BaselineMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

calculate_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

dimension: str | None

finalize_data(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type
type | None

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

type: str

clp_guide

Modules

glotaran.builtin.megacomplexes.clp_guide.clp_guide_megacomplex

clp_guide_megacomplex

Classes

Summary

ClpGuideMegacomplex	Method generated by attrs for class ClpGuideMegacomplex.

ClpGuideMegacomplex

class glotaran.builtin.megacomplexes.clp_guide.clp_guide_megacomplex.**ClpGuideMegacomplex**(*, *label: str*, *dimension: str | None = None*, *type: str = 'clp-guide'*, *target: str*)

Bases: `Megacomplex`

Method generated by attrs for class ClpGuideMegacomplex.

Attributes Summary

type
target
dimension
label

type

ClpGuideMegacomplex.**type**: str

target

ClpGuideMegacomplex.**target**: str

dimension

ClpGuideMegacomplex.**dimension**: str | None

label

ClpGuideMegacomplex.**label**: str

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_dataset_model_type	Get the dataset model type.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate_matrix

ClpGuideMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

ClpGuideMegacomplex.**finalize_data**(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_dataset_model_type

classmethod ClpGuideMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_item_type

classmethod ClpGuideMegacomplex.**get_item_type**() → str

Get the type string.

Return type

str

get_item_type_class

classmethod ClpGuideMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

get_item_types

classmethod ClpGuideMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type

list[str]

Methods Documentation

calculate_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

dimension: str | None

finalize_data(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type

type | None

classmethod get_item_type() → str

Get the type string.

Return type

str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type

list[str]

label: str

target: str

type: str

coherent_artifact

Modules

glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex	This package contains the kinetic megacomplex item.

coherent_artifact_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

CoherentArtifactMegacomplex	Method generated by attrs for class CoherentArtifactMegacomplex.

CoherentArtifactMegacomplex

class glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex.CoherentArtifa

Bases: Megacomplex

Method generated by attrs for class CoherentArtifactMegacomplex.

Attributes Summary

dimension
type
order
width
label

dimension

CoherentArtifactMegacomplex.**dimension: str**

type

CoherentArtifactMegacomplex.**type: str**

order

CoherentArtifactMegacomplex.**order: int**

width

CoherentArtifactMegacomplex.**width: ParameterType | None**

label

CoherentArtifactMegacomplex.**label: str**

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
compartments	
finalize_data	Finalize a dataset.
get_dataset_model_type	Get the dataset model type.
get_irf_parameter	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate_matrix

CoherentArtifactMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

compartments

`CoherentArtifactMegacomplex.`**`compartments`**`()`

finalize_data

`CoherentArtifactMegacomplex.`**`finalize_data`**`(`*dataset_model:* DatasetModel, *dataset:* *Dataset*, *is_full_model:* *bool* = *False*, *as_global:* *bool* = *False*`)`

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_dataset_model_type

classmethod `CoherentArtifactMegacomplex.`**`get_dataset_model_type`**`()` → type | None

Get the dataset model type.

Return type

type | None

get_irf_parameter

`CoherentArtifactMegacomplex.`**`get_irf_parameter`**`(`*irf:* IrfMultiGaussian, *global_index:* *int* | *None*, *global_axis: ArrayLike*`)` → tuple[float, float]

get_item_type

classmethod `CoherentArtifactMegacomplex.`**`get_item_type`**`()` → str

Get the type string.

Return type

str

get_item_type_class

classmethod CoherentArtifactMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

get_item_types

classmethod CoherentArtifactMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type

list[str]

Methods Documentation

calculate_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

compartments()

dimension: str

finalize_data(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type

type | None

get_irf_parameter(*irf: IrfMultiGaussian*, *global_index: int | None*, *global_axis: ArrayLike*) → tuple[float, float]

classmethod get_item_type() → str

Get the type string.

Return type

str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type

list[str]

label: str

order: int

type: str

width: ParameterType | None

damped_oscillation

Modules

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex

damped_oscillation_megacomplex

Functions

Summary

calculate_damped_oscillation_matrix_gau	Calculate the damped oscillation matrix taking into account a gaussian irf
calculate_damped_oscillation_matrix_gau	
*calculate_damped_oscillation_matrix_no_*	
validate_oscillation_parameter	

calculate_damped_oscillation_matrix_gaussian_irf

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**calculate_damped_o**

Calculate the damped oscillation matrix taking into account a gaussian irf

Parameters

- **frequencies** (*np.ndarray*) – an array of frequencies in THz, one per oscillation
- **rates** (*np.ndarray*) – an array of rates, one per oscillation
- **model_axis** (*np.ndarray*) – the model axis (time)
- **center** (*float*) – the center of the gaussian IRF
- **width** (*float*) – the width () parameter of the the IRF
- **shift** (*float*) – a shift parameter per item on the global axis
- **scale** (*float*) – the scale parameter to scale the matrix by

Returns

An array of the real and imaginary part of the oscillation matrix, the shape being (len(model_axis), 2*len(frequencies)), with the first half of the second dimension representing the real part, and the other the imagine part of the oscillation

Return type

np.ndarray

calculate_damped_oscillation_matrix_gaussian_irf_on_index

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**calculate_damped_o**

calculate_damped_oscillation_matrix_no_irf

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**calculate_damped_o**

validate_oscillation_parameter

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**validate_oscillati**

Classes

Summary

DampedOscillationMegacomplex	Method generated by attrs for class DampedOscillationMegacomplex.
OscillationParameterIssue	

DampedOscillationMegacomplex

class glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.DampedOscill

Bases: Megacomplex

Method generated by attrs for class DampedOscillationMegacomplex.

Attributes Summary

dimension
type
labels
frequencies
rates
label

dimension

DampedOscillationMegacomplex.**dimension: str**

type

DampedOscillationMegacomplex.**type: str**

labels

DampedOscillationMegacomplex.**labels: list[str]**

frequencies

DampedOscillationMegacomplex.**frequencies: list[ParameterType]**

rates

DampedOscillationMegacomplex.**rates: list[ParameterType]**

label

DampedOscillationMegacomplex.**label: str**

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_dataset_model_type	Get the dataset model type.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate_matrix

DampedOscillationMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

DampedOscillationMegacomplex.**finalize_data**(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_dataset_model_type

classmethod DampedOscillationMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type
type | None

get_item_type

classmethod DampedOscillationMegacomplex.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod DampedOscillationMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod DampedOscillationMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

calculate_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters
- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

dimension: str

finalize_data(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters
- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

frequencies: list[ParameterType]

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type
type | None

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

labels: list[str]

rates: list[ParameterType]

type: str

OscillationParameterIssue

class glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**OscillationP**

Bases: ItemIssue

Methods Summary

to_string	Get the issue as string.

to_string

OscillationParameterIssue.**to_string**() → str

Get the issue as string.

Returns

- *str*
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

Methods Documentation

to_string() → str

Get the issue as string.

Returns

- *str*
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

decay

Modules

glotaran.builtin.megacomplexes.decay.decay_matrix_gaussian_irf	
glotaran.builtin.megacomplexes.decay.decay_megacomplex	
glotaran.builtin.megacomplexes.decay.decay_parallel_megacomplex	This package contains the decay megacomplex item.
glotaran.builtin.megacomplexes.decay.decay_sequential_megacomplex	This package contains the decay megacomplex item.
glotaran.builtin.megacomplexes.decay.initial_concentration	This package contains the initial concentration item.
glotaran.builtin.megacomplexes.decay.irf	This package contains irf items.
glotaran.builtin.megacomplexes.decay.k_matrix	K-Matrix
glotaran.builtin.megacomplexes.decay.util	

decay_matrix_gaussian_irf

Functions

Summary

calculate_decay_matrix_gaussian_irf	
*calculate_decay_matrix_gaussian_irf_on_*	Calculates a decay matrix with a gaussian irf.

calculate_decay_matrix_gaussian_irf

glotaran.builtin.megacomplexes.decay.decay_matrix_gaussian_irf.**calculate_decay_matrix_gaussian_irf(**

calculate_decay_matrix_gaussian_irf_on_index

glotaran.builtin.megacomplexes.decay.decay_matrix_gaussian_irf.**calculate_decay_matrix_gaussian_irf_**

Calculates a decay matrix with a gaussian irf.

decay_megacomplex

Classes

Summary

DecayDatasetModel	Method generated by attrs for class DecayDatasetModel.
DecayMegacomplex	Method generated by attrs for class DecayMegacomplex.

DecayDatasetModel

class glotaran.builtin.megacomplexes.decay.decay_megacomplex.**DecayDatasetModel**(*, label: str, group: str = 'default', force_index_dependent: bool = False, megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]], megacomplex_scale: list[glotaran.parameter.parameter_group.GroupedParameter | str] | None = None, global_megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None = None, global_megacomplex_scale: list[glotaran.parameter.parameter_group.GroupedParameter | str] | None = None, scale: Parameter | str | None = None, initial_concentration: Initial-Concen-

Bases: *DatasetModel*

Method generated by attrs for class DecayDatasetModel.

Attributes Summary

initial_concentration
irf
group
force_index_dependent
megacomplex
megacomplex_scale
global_megacomplex
global_megacomplex_scale
scale
label

initial_concentration

DecayDatasetModel.**initial_concentration: ModelItemType[*InitialConcentration*] | None**

irf

DecayDatasetModel.**irf: ModelItemType[*Irf*] | None**

group

DecayDatasetModel.**group: str**

force_index_dependent

DecayDatasetModel.**force_index_dependent**: bool

megacomplex

DecayDatasetModel.**megacomplex**: list[ModelItemType[Megacomplex]]

megacomplex_scale

DecayDatasetModel.**megacomplex_scale**: list[ParameterType] | None

global_megacomplex

DecayDatasetModel.**global_megacomplex**: list[ModelItemType[Megacomplex]] | None

global_megacomplex_scale

DecayDatasetModel.**global_megacomplex_scale**: list[ParameterType] | None

scale

DecayDatasetModel.**scale**: ParameterType | None

label

DecayDatasetModel.**label**: str

Methods Summary

Methods Documentation

force_index_dependent: bool

global_megacomplex: list[ModelItemType[Megacomplex]] | None

global_megacomplex_scale: list[ParameterType] | None

group: str

initial_concentration: ModelItemType[*InitialConcentration*] | None

irf: ModelItemType[*Irf*] | None

label: str

megacomplex: list[ModelItemType[Megacomplex]]

megacomplex_scale: list[ParameterType] | None

scale: ParameterType | None

DecayMegacomplex

class glotaran.builtin.megacomplexes.decay.decay_megacomplex.**DecayMegacomplex**(*, *label: str*, *dimension: str = 'time'*, *type: str = 'decay'*, *k_matrix: list[Union[*glotaran.builtin.*str]]*)

Bases: Megacomplex

Method generated by attrs for class DecayMegacomplex.

Attributes Summary

dimension
type
k_matrix
label

dimension

DecayMegacomplex.**dimension**: str

type

DecayMegacomplex.**type**: str

k_matrix

DecayMegacomplex.**k_matrix**: list[ModelItemType[*KMatrix*]]

label

DecayMegacomplex.**label**: str

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_a_matrix	
get_compartments	
get_dataset_model_type	Get the dataset model type.
get_initial_concentration	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
get_k_matrix	

calculate_matrix

DecayMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

DecayMegacomplex.**finalize_data**(*dataset_model:* DatasetModel, *dataset:* *Dataset*, *is_full_model:* *bool* = *False*, *as_global:* *bool* = *False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix

DecayMegacomplex.**get_a_matrix**(*dataset_model:* DatasetModel) → ndarray

get_compartments

DecayMegacomplex.**get_compartments**(*dataset_model:* DatasetModel) → list[str]

get_dataset_model_type

classmethod DecayMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_initial_concentration

DecayMegacomplex.**get_initial_concentration**(*dataset_model:* DatasetModel, *normalized:* *bool* = *True*) → ndarray

get_item_type

classmethod DecayMegacomplex.**get_item_type**() → str

Get the type string.

Return type

str

get_item_type_class

classmethod DecayMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod DecayMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

get_k_matrix

DecayMegacomplex.**get_k_matrix**() → *KMatrix*

Methods Documentation

calculate_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters
- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

dimension: str

finalize_data(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters
- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix(*dataset_model: DatasetModel*) → ndarray

get_compartments(*dataset_model: DatasetModel*) → list[str]

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type
type | None

get_initial_concentration(*dataset_model:* DatasetModel, *normalized:* *bool* = *True*) → ndarray

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type:* *str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

get_k_matrix() → *KMatrix*

k_matrix: list[ModelItemType[*KMatrix*]]

label: str

type: str

decay_parallel_megacomplex

This package contains the decay megacomplex item.

Classes

Summary

DecayDatasetModel	Method generated by attrs for class DecayDatasetModel.
DecayParallelMegacomplex	Method generated by attrs for class DecayParallelMegacomplex.

DecayDatasetModel

class glotaran.builtin.megacomplexes.decay.decay_parallel_megacomplex.**DecayDatasetModel**(*, label: str, group: str = 'default', force_index_dimension: bool = False, megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]], megacomplex_scale: list[glotaran.parameter.parameter.Parameter | str] | None = None, global_megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None = None, global_megacomplex_scale: list[glotaran.parameter.parameter.Parameter | str] | None = None, scale: glotaran.parameter.parameter.Parameter | str | None = None, irf: glotaran.model.irf.Irf | str | None

Bases: *DatasetModel*

Method generated by attrs for class DecayDatasetModel.

Attributes Summary

irf
group
force_index_dependent
megacomplex
megacomplex_scale
global_megacomplex
global_megacomplex_scale
scale
label

irf

DecayDatasetModel.**irf**: *Irf* | str | None

group

DecayDatasetModel.**group**: str

force_index_dependent

DecayDatasetModel.**force_index_dependent**: bool

megacomplex

DecayDatasetModel.**megacomplex**:
list[Union[glotaran.model.megacomplex.Megacomplex, str]]

megacomplex_scale

DecayDatasetModel.**megacomplex_scale**:
list[*glotaran.parameter.parameter.Parameter* | str] | None

global_megacomplex

DecayDatasetModel.**global_megacomplex**:
list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None

global_megacomplex_scale

DecayDatasetModel.**global_megacomplex_scale**:
list[*glotaran.parameter.parameter.Parameter* | str] | None

scale

DecayDatasetModel.**scale**: *Parameter* | str | None

label

DecayDatasetModel.**label**: str

Methods Summary

Methods Documentation

force_index_dependent: bool

global_megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None

global_megacomplex_scale: list[*glotaran.parameter.parameter.Parameter* | str] | None

group: str

irf: *Irf* | str | None

label: str

megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]]

megacomplex_scale: list[*glotaran.parameter.parameter.Parameter* | str] | None

scale: *Parameter* | str | None

DecayParallelMegacomplex

class glotaran.builtin.megacomplexes.decay.decay_parallel_megacomplex.**DecayParallelMegacomplex**(*, label: str, dimension: str = 'time', type: str = 'decay-parallel', compartments: list[str], rates: list[glotaran.model.megacomplex_matrix.MegacomplexMatrix | str])

Bases: Megacomplex

Method generated by attrs for class DecayParallelMegacomplex.

Attributes Summary

dimension
type
compartments
rates
label

dimension

DecayParallelMegacomplex.**dimension**: str

type

DecayParallelMegacomplex.**type**: str

compartments

DecayParallelMegacomplex.**compartments**: list[str]

rates

DecayParallelMegacomplex.**rates**: list[ParameterType]

label

DecayParallelMegacomplex.**label**: str

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_a_matrix	
get_compartments	
get_dataset_model_type	Get the dataset model type.
get_initial_concentration	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
get_k_matrix	

calculate_matrix

DecayParallelMegacomplex.**calculate_matrix**(*dataset_model: DecayDatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

DecayParallelMegacomplex.**finalize_data**(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix

DecayParallelMegacomplex.**get_a_matrix**(*dataset_model: DatasetModel*) → ndarray

get_compartments

DecayParallelMegacomplex.**get_compartments**(*dataset_model: DatasetModel*) → list[str]

get_dataset_model_type

classmethod DecayParallelMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_initial_concentration

DecayParallelMegacomplex.**get_initial_concentration**(*dataset_model:* DatasetModel, *normalized: bool = True*) → ndarray

get_item_type

classmethod DecayParallelMegacomplex.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod DecayParallelMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod DecayParallelMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

get_k_matrix

DecayParallelMegacomplex.**get_k_matrix**() → *KMatrix*

Methods Documentation

calculate_matrix(*dataset_model:* DecayDatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

compartments: list[str]

dimension: str

finalize_data(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix(*dataset_model:* DatasetModel) → ndarray

get_compartments(*dataset_model:* DatasetModel) → list[str]

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type

type | None

get_initial_concentration(*dataset_model:* DatasetModel, *normalized: bool = True*) → ndarray

classmethod get_item_type() → str

Get the type string.

Return type

str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type

list[str]

get_k_matrix() → *KMatrix*

label: str

rates: list[ParameterType]

type: str

decay_sequential_megacomplex

This package contains the decay megacomplex item.

Classes

Summary

DecaySequentialMegacomplex	A Megacomplex with one or more K-Matrices.

DecaySequentialMegacomplex

class glotaran.builtin.megacomplexes.decay.decay_sequential_megacomplex.**DecaySequentialMegacomplex**(

Bases: *DecayParallelMegacomplex*

A Megacomplex with one or more K-Matrices.

Method generated by attrs for class DecaySequentialMegacomplex.

Attributes Summary

type
dimension
compartments
rates
label

type

DecaySequentialMegacomplex.**type**: str

dimension

DecaySequentialMegacomplex.**dimension**: str

compartments

DecaySequentialMegacomplex.**compartments**: list[str]

rates

DecaySequentialMegacomplex.**rates**: list[ParameterType]

label

DecaySequentialMegacomplex.**label**: str

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_a_matrix	
get_compartments	
get_dataset_model_type	Get the dataset model type.
get_initial_concentration	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
get_k_matrix	

calculate_matrix

DecaySequentialMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

DecaySequentialMegacomplex.**finalize_data**(*dataset_model: DatasetModel*, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix

DecaySequentialMegacomplex.**get_a_matrix**(*dataset_model:* DatasetModel) → ndarray

get_compartments

DecaySequentialMegacomplex.**get_compartments**(*dataset_model:* DatasetModel) → list[str]

get_dataset_model_type

classmethod DecaySequentialMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_initial_concentration

DecaySequentialMegacomplex.**get_initial_concentration**(*dataset_model:* DatasetModel, *normalized: bool = True*) → ndarray

get_item_type

classmethod DecaySequentialMegacomplex.**get_item_type**() → str

Get the type string.

Return type

str

get_item_type_class

classmethod DecaySequentialMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

get_item_types

classmethod DecaySequentialMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

get_k_matrix

DecaySequentialMegacomplex.**get_k_matrix**() → *KMatrix*

Methods Documentation

calculate_matrix(*dataset_model:* DatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns
- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

compartments: list[str]

dimension: str

finalize_data(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters
- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_a_matrix(*dataset_model:* DatasetModel) → ndarray

get_compartments(*dataset_model:* DatasetModel) → list[str]

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type
type | None

get_initial_concentration(*dataset_model:* DatasetModel, *normalized: bool = True*) → ndarray

classmethod get_item_type() → str

Get the type string.

Return type

str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type

list[str]

get_k_matrix() → *KMatrix*

label: str

rates: list[ParameterType]

type: str

initial_concentration

This package contains the initial concentration item.

Classes

Summary

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

InitialConcentration

class glotaran.builtin.megacomplexes.decay.initial_concentration.**InitialConcentration**(*, label: str, compartments: list[str], parameters: list[glotaran.para | str], exclude_from_nor list[str] = [])

Bases: ModelItem

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

Method generated by attrs for class InitialConcentration.

Attributes Summary

compartments
parameters
exclude_from_normalize
label

compartments

InitialConcentration.**compartments**: list[str]

parameters

InitialConcentration.**parameters**:
list[*glotaran.parameter.parameter.Parameter* | str]

exclude_from_normalize

InitialConcentration.**exclude_from_normalize**: list[str]

label

InitialConcentration.**label**: str

Methods Summary

normalized

normalized

InitialConcentration.**normalized**() → ndarray

Methods Documentation

compartments: list[str]

exclude_from_normalize: list[str]

label: str

normalized() → ndarray

parameters: list[*glotaran.parameter.parameter.Parameter* | str]

irf

This package contains irf items.

Classes

Summary

Irf	Represents an IRF.
IrfGaussian	Method generated by attrs for class IrfGaussian.
IrfMultiGaussian	Represents a gaussian IRF.
IrfSpectralGaussian	Method generated by attrs for class IrfSpectralGaussian.
IrfSpectralMultiGaussian	Represents a gaussian IRF.

Irf

class glotaran.builtin.megacomplexes.decay.irf.**Irf**(*, *label: str*, *type: str*)

Bases: ModelItemTyped

Represents an IRF.

Method generated by attrs for class Irf.

Attributes Summary

type
label

type

Irf.**type: str**

label

Irf.**label: str**

Methods Summary

get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

get_item_type

classmethod Irf.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod Irf.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod Irf.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

classmethod **get_item_type**() → str

Get the type string.

Return type
str

classmethod **get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod **get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

label: str

type: str

IrfGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfGaussian**(*, *label:* *str*, *scale:* *list*[glotaran.parameter.parameter.Parameter | *str*] | *None* = *None*, *shift:* *list*[glotaran.parameter.parameter.Parameter | *str*] | *None* = *None*, *normalize:* *bool* = *True*, *backsweep:* *bool* = *False*, *backsweep_period:* Parameter | *str* | *None* = *None*, *type:* *str* = *'gaussian'*, *center:* Parameter | *str*, *width:* Parameter | *str*)

Bases: *IrfMultiGaussian*

Method generated by attrs for class IrfGaussian.

Attributes Summary

type
center
width
scale
shift
normalize
backsweep
backsweep_period
label

type

IrfGaussian.**type**: str

center

IrfGaussian.**center**: *Parameter* | str

width

IrfGaussian.**width**: *Parameter* | str

scale

IrfGaussian.**scale**: list[ParameterType] | None

shift

IrfGaussian.**shift**: list[ParameterType] | None

normalize

IrfGaussian.**normalize**: bool

backsweep

IrfGaussian.**backsweep**: bool

backsweep_period

IrfGaussian.**backsweep_period**: ParameterType | None

label

IrfGaussian.**label**: str

Methods Summary

calculate	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
is_index_dependent	
parameter	Returns the properties of the irf with shift applied.

calculate

IrfGaussian.**calculate**(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

get_item_type

classmethod IrfGaussian.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod IrfGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod IrfGaussian.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent

IrfGaussian.**is_index_dependent**()

parameter

IrfGaussian.**parameter**(*global_index: int*, *global_axis: ndarray*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

Methods Documentation

backsweep: bool

backsweep_period: ParameterType | None

calculate(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

center: *Parameter* | str

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent()

label: str

normalize: bool

parameter(*global_index: int*, *global_axis: ndarray*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

scale: list[ParameterType] | None

shift: list[ParameterType] | None

type: str

width: *Parameter* | str

IrfMultiGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfMultiGaussian**(*, *label: str*, *type: str = 'multi-gaussian'*, *center: list[glotaran.parameter.parameter.Parameter | str]*, *width: list[glotaran.parameter.parameter.Parameter | str]*, *scale: list[glotaran.parameter.parameter.Parameter | str] | None = None*, *shift: list[glotaran.parameter.parameter.Parameter | str] | None = None*, *normalize: bool = True*, *backsweep: bool = False*, *backsweep_period: Parameter | str | None = None*)

Bases: *Irf*

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** (*list[glotaran.parameter.parameter.Parameter | str]*) – one or more center of the irf as parameter indices
- **width** (*list[glotaran.parameter.parameter.Parameter | str]*) – one or more widths of the gaussian as parameter index
- **center_dispersion_coefficients** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion_coefficients** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Method generated by attrs for class IrfMultiGaussian.

Attributes Summary

type
center
width
scale
shift
normalize
backsweep
backsweep_period
label

type

IrfMultiGaussian.**type**: str

center

IrfMultiGaussian.**center**: list[*glotaran.parameter.parameter.Parameter* | str]

width

IrfMultiGaussian.**width**: list[*glotaran.parameter.parameter.Parameter* | str]

scale

IrfMultiGaussian.**scale**: list[*glotaran.parameter.parameter.Parameter* | str] | None

shift

IrfMultiGaussian.**shift**: list[*glotaran.parameter.parameter.Parameter* | str] | None

normalize

IrfMultiGaussian.**normalize**: bool

backsweep

IrfMultiGaussian.**backsweep**: bool

backsweep_period

IrfMultiGaussian.**backsweep_period**: *Parameter* | str | None

label

IrfMultiGaussian.**label**: str

Methods Summary

calculate	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
is_index_dependent	
parameter	Returns the properties of the irf with shift applied.

calculate

IrfMultiGaussian.**calculate**(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

get_item_type

classmethod IrfMultiGaussian.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod IrfMultiGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod IrfMultiGaussian.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent

IrfMultiGaussian.**is_index_dependent**()

parameter

IrfMultiGaussian.**parameter**(*global_index: int*, *global_axis: ndarray*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

Methods Documentation

backsweep: bool

backsweep_period: *Parameter* | str | None

calculate(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

center: list[*glotaran.parameter.parameter.Parameter* | str]

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent()

label: str

normalize: bool

parameter(*global_index: int*, *global_axis: ndarray*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

scale: list[*glotaran.parameter.parameter.Parameter* | str] | None

shift: list[*glotaran.parameter.parameter.Parameter* | str] | None

type: str

width: list[*glotaran.parameter.parameter.Parameter* | str]

IrfSpectralGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfSpectralGaussian**(*, *label: str*, *scale: list[glotaran.parameter.parameter.Parame | str] | None = None*, *shift: list[glotaran.parameter.parameter.Parame | str] | None = None*, *normalize: bool = True*, *backsweep: bool = False*, *backsweep_period: Parameter | str | None = None*, *dispersion_center: Parameter | str*, *center_dispersion_coefficients: list[glotaran.parameter.parameter.Parame | str]*, *width_dispersion_coefficients: list[glotaran.parameter.parameter.Parame | str] = _Nothing.NOTHING*, *model_dispersion_with_wavenumber: bool = False*, *type: str = 'spectralgaussian'*, *center: Parameter | str*, *width: Parameter | str*)

Bases: *IrfSpectralMultiGaussian*

Method generated by attrs for class IrfSpectralGaussian.

Attributes Summary

type
center
width
dispersion_center
center_dispersion_coefficients
width_dispersion_coefficients
model_dispersion_with_wavenumber
scale
shift
normalize
backsweep
backsweep_period
label

type

IrfSpectralGaussian.**type**: str

center

IrfSpectralGaussian.**center**: *Parameter* | str

width

IrfSpectralGaussian.**width**: *Parameter* | str

dispersion_center

IrfSpectralGaussian.**dispersion_center: ParameterType**

center_dispersion_coefficients

IrfSpectralGaussian.**center_dispersion_coefficients: list[ParameterType]**

width_dispersion_coefficients

IrfSpectralGaussian.**width_dispersion_coefficients: list[ParameterType]**

model_dispersion_with_wavenumber

IrfSpectralGaussian.**model_dispersion_with_wavenumber: bool**

scale

IrfSpectralGaussian.**scale: list[ParameterType] | None**

shift

IrfSpectralGaussian.**shift: list[ParameterType] | None**

normalize

IrfSpectralGaussian.**normalize: bool**

backsweep

IrfSpectralGaussian.**backsweep: bool**

backsweep_period

IrfSpectralGaussian.**backsweep_period: ParameterType | None**

label

IrfSpectralGaussian.**label**: str

Methods Summary

calculate	
calculate_dispersion	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
is_index_dependent	
parameter	Returns the properties of the irf with shift and dispersion applied.

calculate

IrfSpectralGaussian.**calculate**(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

calculate_dispersion

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

get_item_type

classmethod IrfSpectralGaussian.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod IrfSpectralGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod IrfSpectralGaussian.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent

IrfSpectralGaussian.**is_index_dependent**()

parameter

IrfSpectralGaussian.**parameter**(*global_index: int*, *global_axis: ndarray*)

Returns the properties of the irf with shift and dispersion applied.

Methods Documentation

backsweep: bool

backsweep_period: ParameterType | None

calculate(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

calculate_dispersion(*axis*)

center: *Parameter* | str

center_dispersion_coefficients: list[ParameterType]

dispersion_center: ParameterType

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent()

label: str

model_dispersion_with_wavenumber: bool

normalize: bool

parameter(*global_index: int*, *global_axis: ndarray*)

Returns the properties of the irf with shift and dispersion applied.

scale: list[ParameterType] | None

shift: list[ParameterType] | None

type: str

width: *Parameter* | str

width_dispersion_coefficients: list[ParameterType]

IrfSpectralMultiGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfSpectralMultiGaussian**(*, *label: str*, *center: list[glotaran.parameter.parameter.P | str]*, *width: list[glotaran.parameter.parameter.P | str]*, *scale: list[glotaran.parameter.parameter.P | str] | None = None*, *shift: list[glotaran.parameter.parameter.P | str] | None = None*, *normalize: bool = True*, *backsweep: bool = False*, *backsweep_period: Parameter | str | None = None*, *type: str = 'spectral-multi-gaussian'*, *dispersion_center: Parameter | str*, *center_dispersion_coefficients: list[glotaran.parameter.parameter.P | str]*, *width_dispersion_coefficients: list[glotaran.parameter.parameter.P | str] = _Nothing.NOTHING*, *model_dispersion_with_wavenumb bool = False*)

Bases: *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_coefficients** (*list[glotaran.parameter.parameter.Parameter | str]*) – list of parameters with polynomial coefficients describing the dispersion of the irf center location. None for no dispersion.
- **width_dispersion_coefficients** (*list[glotaran.parameter.parameter.Parameter | str]*) – list of parameters with polynomial coefficients describing the dispersion of the width of the irf. None for no dispersion.

Method generated by attrs for class IrfSpectralMultiGaussian.

Attributes Summary

type
dispersion_center
center_dispersion_coefficients
width_dispersion_coefficients
model_dispersion_with_wavenumber
center
width
scale
shift
normalize
backsweep
backsweep_period
label

type

IrfSpectralMultiGaussian.**type**: str

dispersion_center

IrfSpectralMultiGaussian.**dispersion_center**: *Parameter* | str

center_dispersion_coefficients

IrfSpectralMultiGaussian.**center_dispersion_coefficients**:
list[*glotaran.parameter.parameter.Parameter* | str]

width_dispersion_coefficients

IrfSpectralMultiGaussian.**width_dispersion_coefficients**:
list[*glotaran.parameter.parameter.Parameter* | str]

model_dispersion_with_wavenumber

IrfSpectralMultiGaussian.**model_dispersion_with_wavenumber**: bool

center

IrfSpectralMultiGaussian.**center**: list[ParameterType]

width

IrfSpectralMultiGaussian.**width**: list[ParameterType]

scale

IrfSpectralMultiGaussian.**scale**: list[ParameterType] | None

shift

IrfSpectralMultiGaussian.**shift**: list[ParameterType] | None

normalize

IrfSpectralMultiGaussian.**normalize**: bool

backsweep

IrfSpectralMultiGaussian.**backsweep**: bool

backsweep_period

IrfSpectralMultiGaussian.**backsweep_period**: ParameterType | None

label

IrfSpectralMultiGaussian.**label**: str

Methods Summary

calculate	
calculate_dispersion	
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
is_index_dependent	
parameter	Returns the properties of the irf with shift and dispersion applied.

calculate

IrfSpectralMultiGaussian.**calculate**(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

calculate_dispersion

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

get_item_type

classmethod IrfSpectralMultiGaussian.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod IrfSpectralMultiGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod IrfSpectralMultiGaussian.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent

IrfSpectralMultiGaussian.**is_index_dependent**()

parameter

IrfSpectralMultiGaussian.**parameter**(*global_index: int*, *global_axis: ndarray*)

Returns the properties of the irf with shift and dispersion applied.

Methods Documentation

backsweep: bool

backsweep_period: ParameterType | None

calculate(*index: int*, *global_axis: ndarray*, *model_axis: ndarray*) → ndarray

calculate_dispersion(*axis*)

center: list[ParameterType]

center_dispersion_coefficients: list[*glotaran.parameter.parameter.Parameter* | str]

dispersion_center: *Parameter* | str

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

is_index_dependent()

label: str

model_dispersion_with_wavenumber: bool

normalize: bool

parameter(*global_index: int*, *global_axis: ndarray*)

Returns the properties of the irf with shift and dispersion applied.

scale: list[ParameterType] | None

shift: list[ParameterType] | None

type: str

width: list[ParameterType]

width_dispersion_coefficients: list[*glotaran.parameter.parameter.Parameter* | str]

k_matrix

K-Matrix

Functions

Summary

calculate_gamma

calculate_gamma

glotaran.builtin.megacomplexes.decay.k_matrix.**calculate_gamma**(*eigenvectors: ndarray*, *initial_concentration: ndarray*) → ndarray

Classes

Summary

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

KMatrix

class glotaran.builtin.megacomplexes.decay.k_matrix.**KMatrix**(*, *label: str*, *matrix: dict[tuple[str, str], glotaran.parameter.parameter.Parameter | str]*)

Bases: ModelItem

A K-Matrix represents a first order differental system.

Method generated by attrs for class KMatrix.

Attributes Summary

matrix
label

matrix

KMatrix.**matrix: dict[tuple[str, str], *glotaran.parameter.parameter.Parameter* | str]**

label

KMatrix.**label**: str

Methods Summary

a_matrix	The A matrix of the KMatrix.
a_matrix_as_markdown	Returns the A Matrix as markdown formatted table.
a_matrix_general	The A matrix of the KMatrix for a general model.
a_matrix_sequential	The A matrix of the KMatrix for a sequential model.
combine	Creates a combined matrix.
eigen	Returns the eigenvalues and eigenvectors of the k matrix.
empty	Creates an empty K-Matrix.
full	The full representation of the KMatrix as numpy array.
involved_compartments	A list of all compartments in the Matrix.
is_sequential	Returns true in the KMatrix represents an unibranched model.
matrix_as_markdown	Returns the KMatrix as markdown formatted table.
rates	The resulting rates of the matrix.
reduced	The reduced representation of the KMatrix as numpy array.

a_matrix

KMatrix.**a_matrix**(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The A matrix of the KMatrix.

Parameters

initial_concentration – The initial concentration.

a_matrix_as_markdown

KMatrix.**a_matrix_as_markdown**(*compartments: list[str]*, *initial_concentration: ndarray*) → *MarkdownStr*

Returns the A Matrix as markdown formatted table.

Parameters

initial_concentration – The initial concentration.

a_matrix_general

KMatrix.**a_matrix_general**(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The A matrix of the KMatrix for a general model.

Parameters

initial_concentration – The initial concentration.

a_matrix_sequential

KMatrix.**a_matrix_sequential**(*compartments: list[str]*) → ndarray

The A matrix of the KMatrix for a sequential model.

Parameters

initial_concentration – The initial concentration.

combine

KMatrix.**combine**(*k_matrix:* KMatrix) → *KMatrix*

Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.

Parameters

k_matrix – KMatrix to combine with.

Returns

The combined KMatrix.

Return type

combined

eigen

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

full

KMatrix.**full**(*compartments: list[str]*) → 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_sequential

KMatrix.**is_sequential**(*compartments: list[str]*, *initial_concentration: ndarray*) → bool

Returns true in the KMatrix represents an unibranched model.

Parameters

initial_concentration – The initial concentration.

matrix_as_markdown

KMatrix.**matrix_as_markdown**(*compartments: list[str] | None = None*, *fill_parameters: bool = False*) → *MarkdownStr*

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.

rates

KMatrix.**rates**(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The resulting rates of the matrix.

By definition, the eigenvalues of the compartmental model are negative and the rates are the negatives of the eigenvalues, thus the eigenvalues need to be multiplied with -1 to get rates with the correct sign.

Parameters

- **compartments** (*list[str]*) – Names of compartment used to order the matrix.
- **initial_concentration** (*np.ndarray*) – The initial concentration.

reduced

KMatrix.**reduced**(*compartments: list[str]*) → ndarray

The reduced representation of the KMatrix as numpy array.

Parameters

compartments – The compartment order.

Methods Documentation

a_matrix(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The A matrix of the KMatrix.

Parameters

initial_concentration – The initial concentration.

a_matrix_as_markdown(*compartments: list[str]*, *initial_concentration: ndarray*) → *MarkdownStr*

Returns the A Matrix as markdown formatted table.

Parameters

initial_concentration – The initial concentration.

a_matrix_general(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The A matrix of the KMatrix for a general model.

Parameters

initial_concentration – The initial concentration.

a_matrix_sequential(*compartments: list[str]*) → ndarray

The A matrix of the KMatrix for a sequential model.

Parameters

initial_concentration – The initial concentration.

combine(*k_matrix:* KMatrix) → *KMatrix*

Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.

Parameters

k_matrix – KMatrix to combine with.

Returns

The combined KMatrix.

Return type

combined

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

full(*compartments: list[str]*) → 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_sequential(*compartments: list[str]*, *initial_concentration: ndarray*) → bool

Returns true in the KMatrix represents an unibranched model.

Parameters

initial_concentration – The initial concentration.

label: str

matrix: dict[tuple[str, str], *glotaran.parameter.parameter.Parameter* | str]

matrix_as_markdown(*compartments: list[str] | None = None*, *fill_parameters: bool = False*) → *MarkdownStr*

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.

rates(*compartments: list[str]*, *initial_concentration: ndarray*) → ndarray

The resulting rates of the matrix.

By definition, the eigenvalues of the compartmental model are negative and the rates are the negatives of the eigenvalues, thus the eigenvalues need to be multiplied with -1 to get rates with the correct sign.

Parameters

- **compartments** (*list[str]*) – Names of compartment used to order the matrix.
- **initial_concentration** (*np.ndarray*) – The initial concentration.

reduced(*compartments: list[str]*) → ndarray

The reduced representation of the KMatrix as numpy array.

Parameters

compartments – The compartment order.

util

Functions

Summary

calculate_decay_matrix_no_irf	
calculate_matrix	
collect_megacomplexes	
decay_matrix_implementation_index_deper	
decay_matrix_implementation_index_indep	
finalize_data	
index_dependent	Determine if a dataset_model is index dependent.
retrieve_decay_associated_data	
retrieve_initial_concentration	
retrieve_irf	
retrieve_species_associated_data	

calculate_decay_matrix_no_irf

glotaran.builtin.megacomplexes.decay.util.**calculate_decay_matrix_no_irf**(*matrix*, *rates*, *times*)

calculate_matrix

glotaran.builtin.megacomplexes.decay.util.**calculate_matrix**(*megacomplex: Megacomplex*, *dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

collect_megacomplexes

glotaran.builtin.megacomplexes.decay.util.**collect_megacomplexes**(*dataset_model:* DatasetModel, *as_global: bool*) → list[glotaran.model.megacomplex.Megacomp

decay_matrix_implementation_index_dependent

glotaran.builtin.megacomplexes.decay.util.**decay_matrix_implementation_index_dependent**(*matrix:* ndarray, *rates:* ndarray, *global_axis:* ndarray, *model_axis:* ndarray, *dataset_model:* DatasetModel)

decay_matrix_implementation_index_independent

glotaran.builtin.megacomplexes.decay.util.**decay_matrix_implementation_index_independent**(*matrix:* ndarray, *rates:* ndarray, *global_axis:* ndarray, *model_axis:* ndarray, *dataset_mode* DatasetModel)

finalize_data

glotaran.builtin.megacomplexes.decay.util.**finalize_data**(*dataset_model:* DatasetModel, *dataset:* *Dataset*, *is_full_model:* *bool* = *False*, *as_global:* *bool* = *False*)

index_dependent

glotaran.builtin.megacomplexes.decay.util.**index_dependent**(*dataset_model:* DatasetModel) → bool

Determine if a dataset_model is index dependent.

Parameters
dataset_model (DatasetModel) – A dataset model instance.

Returns
Returns True if the dataset_model has an IRF that is index dependent (e.g. has dispersion).

Return type
bool

retrieve_decay_associated_data

glotaran.builtin.megacomplexes.decay.util.**retrieve_decay_associated_data**(*megacomplex:* *Megacomplex*, *dataset_model:* DatasetModel, *dataset:* *Dataset*, *global_dimension:* *str*, *name:* *str*)

retrieve_initial_concentration

glotaran.builtin.megacomplexes.decay.util.**retrieve_initial_concentration**(*dataset_model:* DatasetModel, *dataset:* *Dataset*, *species_dimension:* *str*)

retrieve_irf

glotaran.builtin.megacomplexes.decay.util.**retrieve_irf**(*dataset_model:* DatasetModel, *dataset:* *Dataset*, *global_dimension:* *str*)

retrieve_species_associated_data

glotaran.builtin.megacomplexes.decay.util.**retrieve_species_associated_data**(*dataset_model:* DatasetModel, *dataset:* *Dataset*, *species:* *list[str]*, *species_dimension:* *str*, *global_dimension:* *str*, *name:* *str*, *is_full_model:* *bool*, *as_global:* *bool*)

spectral

Modules

glotaran.builtin.megacomplexes.spectral.shape	This package contains the spectral shape item.
glotaran.builtin.megacomplexes.spectral.spectral_megacomplex	

shape

This package contains the spectral shape item.

Classes

Summary

SpectralShape	Method generated by attrs for class SpectralShape.
SpectralShapeGaussian	A Gaussian spectral shape
SpectralShapeOne	A constant spectral shape with value 1
SpectralShapeSkewedGaussian	A skewed Gaussian spectral shape
SpectralShapeZero	A constant spectral shape with value 0

SpectralShape

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShape**(*, *label: str*, *type: str*)

Bases: ModelItemTyped

Method generated by attrs for class SpectralShape.

Attributes Summary

type
label

type

SpectralShape.**type: str**

label

SpectralShape.**label: str**

Methods Summary

get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

get_item_type

classmethod SpectralShape.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod SpectralShape.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod SpectralShape.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

type: str

SpectralShapeGaussian

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeGaussian**(*, *label: str*, *type: str = 'gaussian'*, *amplitude: Parameter | str | None = None*, *location: Parameter | str*, *width: Parameter | str*)

Bases: *SpectralShape*

A Gaussian spectral shape

Method generated by attrs for class SpectralShapeGaussian.

Attributes Summary

type
amplitude
location
width
label

type

SpectralShapeGaussian.**type**: str

amplitude

SpectralShapeGaussian.**amplitude**: *Parameter* | str | None

location

SpectralShapeGaussian.**location**: *Parameter* | str

width

SpectralShapeGaussian.**width**: *Parameter* | str

label

SpectralShapeGaussian.**label**: str

Methods Summary

calculate	Calculate a normal Gaussian shape for a given `axis`.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate

SpectralShapeGaussian.**calculate**(*axis: ndarray*) → ndarray

Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp\left(-\frac{\log(2)\,(2(x - x_0))^2}{\Delta^2}\right)$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters
axis (*np.ndarray*) – The axis to calculate the shape for.
Returns
An array representing a Gaussian shape.
Return type
np.ndarray

get_item_type

classmethod SpectralShapeGaussian.**get_item_type**() → str

Get the type string.
Return type
str

get_item_type_class

classmethod SpectralShapeGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.
Parameters
item_type (*str*) – The type string.
Return type
Type

get_item_types

classmethod SpectralShapeGaussian.**get_item_types**() → list[str]

Get all type strings.
Return type
list[str]

Methods Documentation

amplitude: *Parameter* | str | None

calculate(*axis: ndarray*) → ndarray

Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp\left(-\frac{\log(2)\,(2(x - x_0))^2}{\Delta^2}\right)$$

The parameters of the equation represent the following attributes of the shape:
- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters
axis (*np.ndarray*) – The axis to calculate the shape for.
Returns
An array representing a Gaussian shape.
Return type
np.ndarray

classmethod get_item_type() → str

Get the type string.
Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.
Parameters
item_type (*str*) – The type string.
Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.
Return type
list[str]

label: str

location: *Parameter* | str

type: str

width: *Parameter* | str

SpectralShapeOne

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeOne**(*, *label: str*, *type: str = 'one'*)

Bases: *SpectralShape*

A constant spectral shape with value 1

Method generated by attrs for class SpectralShapeOne.

Attributes Summary

type
label

type

SpectralShapeOne.**type: str**

label

SpectralShapeOne.**label: str**

Methods Summary

calculate	calculate calculates the shape.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate

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

calculate calculates the shape.

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

Returns
shape

Return type
numpy.ndarray

get_item_type

classmethod SpectralShapeOne.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod SpectralShapeOne.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod SpectralShapeOne.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

calculate(*axis: ndarray*) → ndarray

calculate calculates the shape.

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

Returns
shape

Return type
numpy.ndarray

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

type: str

SpectralShapeSkewedGaussian

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeSkewedGaussian**(*, *label: str*, *amplitude: Parameter | str | None = None*, *location: Parameter | str*, *width: Parameter | str*, *type: str = 'skewed-gaussian'*, *skewness: Parameter | str*)

Bases: *SpectralShapeGaussian*

A skewed Gaussian spectral shape

Method generated by attrs for class SpectralShapeSkewedGaussian.

Attributes Summary

type
skewness
amplitude
location
width
label

type

SpectralShapeSkewedGaussian.**type**: str

skewness

SpectralShapeSkewedGaussian.**skewness**: *Parameter* | str

amplitude

SpectralShapeSkewedGaussian.**amplitude**: ParameterType | None

location

SpectralShapeSkewedGaussian.**location**: ParameterType

width

SpectralShapeSkewedGaussian.**width**: ParameterType

label

SpectralShapeSkewedGaussian.**label**: str

Methods Summary

calculate	Calculate the skewed Gaussian shape for `axis`.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate

`SpectralShapeSkewedGaussian.`**`calculate`**`(`*axis: ndarray*`)` → ndarray

Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2) \log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b \to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in *SpectralShapeGaussian.calculate()*.

Parameters
`axis` (*np.ndarray*) – The axis to calculate the shape for.

Returns
An array representing a skewed Gaussian shape.

Return type
np.ndarray

get_item_type

`classmethod` `SpectralShapeSkewedGaussian.`**`get_item_type`**`()` → str

Get the type string.

Return type
str

get_item_type_class

classmethod SpectralShapeSkewedGaussian.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod SpectralShapeSkewedGaussian.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

amplitude: ParameterType | None

calculate(*axis: ndarray*) → ndarray

Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2) \log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b \to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in *SpectralShapeGaussian.calculate()*.

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

Returns
An array representing a skewed Gaussian shape.

Return type
np.ndarray

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

location: ParameterType

skewness: *Parameter* | str

type: str

width: ParameterType

SpectralShapeZero

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeZero**(*, *label: str*, *type: str = 'zero'*)

Bases: *SpectralShape*

A constant spectral shape with value 0

Method generated by attrs for class SpectralShapeZero.

Attributes Summary

type
label

type

SpectralShapeZero.**type: str**

label

SpectralShapeZero.**label: str**

Methods Summary

calculate	calculate calculates the shape.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate

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

calculate calculates the shape.

Only works after calling `fill`.

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

Returns
shape

Return type
numpy.ndarray

get_item_type

classmethod SpectralShapeZero.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod SpectralShapeZero.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod SpectralShapeZero.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

calculate(*axis: ndarray*) → ndarray

calculate calculates the shape.

Only works after calling `fill`.

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

Returns
shape

Return type
numpy.ndarray

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

label: str

type: str

spectral_megacomplex

Classes

Summary

SpectralDatasetModel	Method generated by attrs for class SpectralDatasetModel.
SpectralMegacomplex	Method generated by attrs for class SpectralMegacomplex.

SpectralDatasetModel

class glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.**SpectralDatasetModel**(*, label: str, group: str = 'default', force_index_d bool = False, megacomplex: list[Union[glo str]], megacomplex_scale: list[glotaran.p | str] | None = None, global_megac list[Union[glo str]] | None = None, global_megac list[glotaran.p | str] | None = None, scale: Parameter | str | None = None, spectral_axis_inve bool = False, spec-

Bases: *DatasetModel*

Method generated by attrs for class SpectralDatasetModel.

Attributes Summary

spectral_axis_inverted
spectral_axis_scale
group
force_index_dependent
megacomplex
megacomplex_scale
global_megacomplex
global_megacomplex_scale
scale
label

spectral_axis_inverted

SpectralDatasetModel.**spectral_axis_inverted: bool**

spectral_axis_scale

SpectralDatasetModel.**spectral_axis_scale: float**

group

SpectralDatasetModel.**group: str**

force_index_dependent

SpectralDatasetModel.**force_index_dependent: bool**

megacomplex

SpectralDatasetModel.**megacomplex: list[ModelItemType[Megacomplex]]**

megacomplex_scale

SpectralDatasetModel.**megacomplex_scale: list[ParameterType] | None**

global_megacomplex

SpectralDatasetModel.**global_megacomplex: list[ModelItemType[Megacomplex]] | None**

global_megacomplex_scale

SpectralDatasetModel.**global_megacomplex_scale: list[ParameterType] | None**

scale

SpectralDatasetModel.**scale: ParameterType | None**

label

SpectralDatasetModel.**label: str**

Methods Summary

Methods Documentation

force_index_dependent: bool

global_megacomplex: list[ModelItemType[Megacomplex]] | None

global_megacomplex_scale: list[ParameterType] | None

group: str

label: str

megacomplex: list[ModelItemType[Megacomplex]]

megacomplex_scale: list[ParameterType] | None

scale: ParameterType | None

spectral_axis_inverted: bool

spectral_axis_scale: float

SpectralMegacomplex

class glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.**SpectralMegacomplex**(*, *label: str*, *dimension: str* = *'spectral'*, *type: str* = *'spectral'*, *shape: dict[str, Union[*glotaran *str]]*)

Bases: Megacomplex

Method generated by attrs for class SpectralMegacomplex.

Attributes Summary

dimension
type
shape
label

dimension

SpectralMegacomplex.**dimension: str**

type

SpectralMegacomplex.**type: str**

shape

SpectralMegacomplex.**shape: dict[str, ModelItemType[*SpectralShape*]]**

label

SpectralMegacomplex.**label: str**

Methods Summary

calculate_matrix	Calculate the megacomplex matrix.
finalize_data	Finalize a dataset.
get_dataset_model_type	Get the dataset model type.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

calculate_matrix

SpectralMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

finalize_data

SpectralMegacomplex.**finalize_data**(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

get_dataset_model_type

classmethod SpectralMegacomplex.**get_dataset_model_type**() → type | None

Get the dataset model type.

Return type

type | None

get_item_type

classmethod SpectralMegacomplex.**get_item_type**() → str

Get the type string.

Return type

str

get_item_type_class

classmethod SpectralMegacomplex.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

get_item_types

classmethod SpectralMegacomplex.**get_item_types**() → list[str]

Get all type strings.

Return type

list[str]

Methods Documentation

calculate_matrix(*dataset_model:* DatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, ***kwargs*)

Calculate the megacomplex matrix.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- ****kwargs** – Additional arguments.

Returns

- *tuple[list[str], ArrayLike]* – The clp labels and the matrix.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

dimension: str

finalize_data(*dataset_model:* DatasetModel, *dataset: Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

Finalize a dataset.

Parameters

- **dataset_model** (*DatasetModel*) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.
- **is_full_model** (*bool*) – Whether the model is a full model.
- **as_global** (*bool*) – Whether megacomplex is calculated as global megacomplex.

classmethod get_dataset_model_type() → type | None

Get the dataset model type.

Return type

type | None

classmethod get_item_type() → str

Get the type string.

Return type

str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters

item_type (*str*) – The type string.

Return type

Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type

list[str]

label: str

shape: dict[str, ModelItemType[*SpectralShape*]]

type: str

16.1.3 cli

Modules

glotaran.cli.commands	
glotaran.cli.main	The glotaran CLI main function.

commands

Modules

glotaran.cli.commands.explore
glotaran.cli.commands.export
glotaran.cli.commands.optimize
glotaran.cli.commands.pluginlist
glotaran.cli.commands.print
glotaran.cli.commands.util
glotaran.cli.commands.validate

explore

Functions

Summary

export	Exports data from netCDF4 to ascii.

export

glotaran.cli.commands.explore.**export**(*filename: str*, *select*, *out: str*, *name: str*)

Exports data from netCDF4 to ascii.

export

optimize

Functions

Summary

optimize_cmd	Optimizes a model.

optimize_cmd

glotaran.cli.commands.optimize.**optimize_cmd**(*dataformat: str*, *data: list[str]*, *out: str*, *outformat: str*, *nfev: int*, *nnls: bool*, *yes: bool*, *parameters_file: str*, *model_file: str*, *scheme_file: str*)

Optimizes a model. e.g.: glotaran optimize –

pluginlist

Functions

Summary

plugin_list_cmd	Prints a list of installed plugins.

plugin_list_cmd

glotaran.cli.commands.pluginlist.**plugin_list_cmd**()

Prints a list of installed plugins.

print

Functions

Summary

print_cmd	Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

print_cmd

glotaran.cli.commands.print.**print_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

util

Functions

Summary

load_dataset_file	
load_model_file	
load_parameter_file	
load_scheme_file	
project_io_list_supporting_plugins	List all project-io plugin that implement `method_name`.
select_data	
select_name	
signature_analysis	
write_data	

load_dataset_file

glotaran.cli.commands.util.**load_dataset_file**(*filename*, *fmt=None*, *verbose=False*)

load_model_file

glotaran.cli.commands.util.**load_model_file**(*filename*, *verbose=False*)

load_parameter_file

glotaran.cli.commands.util.**load_parameter_file**(*filename*, *fmt=None*, *verbose=False*)

load_scheme_file

glotaran.cli.commands.util.**load_scheme_file**(*filename*, *verbose=False*)

project_io_list_supporting_plugins

glotaran.cli.commands.util.**project_io_list_supporting_plugins**(*method_name: str*, *block_list: Iterable[str] | None = None*) → Iterable[str]

List all project-io plugin that implement `method_name`.

Parameters

- **method_name** (*str*) – Name of the method which should be supported.
- **block_list** (*Iterable[str]*) – Iterable of plugin names which should be omitted.

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: ParamType

Attributes Summary

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

arity

ValOrRangeOrList.**arity: t.ClassVar[int] = 1**

envvar_list_splitter

ValOrRangeOrList.**envvar_list_splitter: t.ClassVar[t.Optional[str]] = 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: t.ClassVar[bool] = False**

name

ValOrRangeOrList.**name: str = 'number or range or list'**

the descriptive name of this type

Methods Summary

convert	Convert the value to the correct type.
fail	Helper method to fail with an invalid value message.
get_metavar	Returns the metavar default for this param if it provides one.
get_missing_message	Optionally might return extra information about a missing parameter.
shell_complete	Return a list of `CompletionItem` objects for the incomplete value.
split_envvar_value	Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.
to_info_dict	Gather information that could be useful for a tool generating user-facing documentation.

convert

ValOrRangeOrList.**convert**(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

fail

ValOrRangeOrList.**fail**(*message: str*, *param:* Parameter | *None* = *None*, *ctx: Context* | *None* = *None*) → t.NoReturn

Helper method to fail with an invalid value message.

get_metavar

ValOrRangeOrList.**get_metavar**(*param:* Parameter) → str | None

Returns the metavar default for this param if it provides one.

get_missing_message

ValOrRangeOrList.**get_missing_message**(*param:* Parameter) → str | None

Optionally might return extra information about a missing parameter.

New in version 2.0.

shell_complete

ValOrRangeOrList.**shell_complete**(*ctx: Context*, *param:* Parameter, *incomplete: str*) → List[CompletionItem]

Return a list of `CompletionItem` objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value

ValOrRangeOrList.**split_envvar_value**(*rv: str*) → Sequence[str]

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.

to_info_dict

ValOrRangeOrList.**to_info_dict**() → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

Methods Documentation

arity: t.ClassVar[int] = 1

convert(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

envvar_list_splitter: t.ClassVar[t.Optional[str]] = 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: str*, *param:* Parameter | *None* = *None*, *ctx: Context* | *None* = *None*) → t.NoReturn

Helper method to fail with an invalid value message.

get_metavar(*param:* Parameter) → str | None

Returns the metavar default for this param if it provides one.

get_missing_message(*param:* Parameter) → str | None

Optionally might return extra information about a missing parameter.

New in version 2.0.

is_composite: t.ClassVar[bool] = False

name: str = 'number or range or list'

the descriptive name of this type

shell_complete(*ctx: Context*, *param:* Parameter, *incomplete: str*) → List[CompletionItem]

Return a list of `CompletionItem` objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value(*rv: str*) → Sequence[str]

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.

to_info_dict() → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

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

glotaran.cli.**main = <Cli main>**

The glotaran CLI main function.

16.1.4 deprecation

Deprecation helpers and place to put deprecated implementations till removing.

Modules

glotaran.deprecation.deprecation_utils	Helper functions to give deprecation warnings.
glotaran.deprecation.modules	Package containing deprecated implementations which were removed.

deprecation_utils

Helper functions to give deprecation warnings.

Functions

Summary

check_overdue	Check if a deprecation is overdue for removal.
check_qualnames_in_tests	Test that qualnames import path exists when running tests.
deprecate	Decorate a function, method or class to deprecate it.
deprecate_dict_entry	Replace dict entry inplace and warn about usage change, if present in the dict.
deprecate_module_attribute	Import and return and anttribute from the new location.
deprecate_submodule	Create a module at runtime which retrieves attributes from new module.
glotaran_version	Version of the distribution.
module_attribute	Import and return the attribute (e.g.
parse_version	Parse version string to tuple of three ints for comparison.
raise_deprecation_error	Raise `GlotaranDeprectedApiError` error, with formatted message.
warn_deprecated	Raise deprecation warning with change information.

check_overdue

glotaran.deprecation.deprecation_utils.**check_overdue**(*deprecated_qual_name_usage: str*, *to_be_removed_in_version: str*) → None

Check if a deprecation is overdue for removal.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Raises

OverDueDeprecation – If the current version is greater or equal to `to_be_removed_in_version`.

check_qualnames_in_tests

glotaran.deprecation.deprecation_utils.**check_qualnames_in_tests**(*qual_names: Sequence[str]*, *importable_indices: Sequence[int]*)

Test that qualnames import path exists when running tests.

All deprecations should be tested anyway in order to get the proper errors when a deprecation is overdue. This helperfunction also helps to ensure that at least the import paths (`qual_names`) of the old and new usage exist.

Parameters

- **qual_names** (*Sequence[str]*) – Sequence of fully qualified module attribute names, optionally with call arguments.
- **importable_indices** (*Sequence[int]*) – Indices of corresponding to `qual_names` indicating how to slice each `qual_name` split at `.`, for the import and attribute checking.

See also:

warn_deprecated, *deprecate*

deprecate

glotaran.deprecation.deprecation_utils.**deprecate**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*, *has_glotaran_replacement: bool = True*, *importable_indices: tuple[int, int] = (1, 1)*) → Callable[[DecoratedCallable], DecoratedCallable]

Decorate a function, method or class to deprecate it.

This raises deprecation warning with old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **has_glotaran_replacement** (*bool*) – Whether or not this functionality has a replacement in core pyglotaran. This will be mapped to the second entry of `check_qualnames` in *warn_deprecated()*.
- **importable_indices** (*Sequence[int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use

importable_indices=(2, 1), this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`. Default

Returns
Original function or class throwing a Deprecation warning when used.

Return type
DecoratedCallable

Raises
OverDueDeprecation – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

warn_deprecated, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` was deprecated and the usage of `load_model` was promoted instead.

Listing 1: glotaran/deprecation/modules/glotaran_root.py

```
@deprecate(
    deprecated_qualname_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
    new_qualname_usage="glotaran.io.load_model(model_path)",
    to_be_removed_in_version="0.6.0",
)
def read_parameters_from_yaml_file(model_path: str):
    return load_model(model_path)
```

deprecate_dict_entry

glotaran.deprecation.deprecation_utils.**deprecate_dict_entry**(*, *dict_to_check: MutableMapping[Hashable, Any]*, *deprecated_usage: str*, *new_usage: str*, *to_be_removed_in_version: str*, *swap_keys: tuple[collections.abc.Hashable, collections.abc.Hashable] | None = None*, *replace_rules: tuple[collections.abc.Mapping[collections.abc.Hashable, Any], collections.abc.Mapping[collections.abc.Hashable, Any]] | None = None*, *stacklevel: int = 3*) → None

Replace dict entry inplace and warn about usage change, if present in the dict.

Parameters

- **dict_to_check** (*MutableMapping[Hashable, Any]*) – Dict which should be checked.
- **deprecated_usage** (*str*) – Old usage to inform user (only used in warning).
- **new_usage** (*str*) – New usage to inform user (only used in warning).
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **swap_keys** (*tuple[Hashable, Hashable]*) – (old_key, new_key), `dict_to_check[new_key]` will be assigned the value `dict_to_check[old_key]` and `old_key` will be removed from the dict. by default None
- **replace_rules** (*Mapping[Hashable, tuple[Any, Any]]*) – ({old_key: old_value}, {new_key: new_value}), If `dict_to_check[old_key]` has the value `old_value`, `dict_to_check[new_key]` it will be set to `new_value`. `old_key` will be removed from the dict if `old_key` and `new_key` aren't equal. by default None
- **stacklevel** (*int*) – Stack at which the warning should be shown as raise. , by default 3

Raises

- **ValueError** – If both `swap_keys` and `replace_rules` are None (default) or not None.
- **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

warn_deprecated

Notes

To prevent confusion exactly one of `replace_rules` and `swap_keys` needs to be passed.

Examples

For readability sake the warnings won't be shown in the examples.

Swapping key names:

```
>>> dict_to_check = {"foo": 123}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="foo",
        new_usage="bar",
        to_be_removed_in_version="0.6.0",
        swap_keys=("foo", "bar")
    )
```

(continues on next page)

(continued from previous page)

```
>>> dict_to_check
{"bar": 123}
```

Changing values:

```
>>> dict_to_check = {"foo": 123}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="foo: 123",
        new_usage="foo: 123.0",
        to_be_removed_in_version="0.6.0",
        replace_rules=({"foo": 123}, {"foo": 123.0})
    )
>>> dict_to_check
{"foo": 123.0}
```

Swapping key names AND changing values:

```
>>> dict_to_check = {"type": "kinetic-spectrum"}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="type: kinectic-spectrum",
        new_usage="default_megacomplex: decay",
        to_be_removed_in_version="0.6.0",
        replace_rules=({"type": "kinetic-spectrum"}, {"default_megacomplex
↪": "decay"})
    )
>>> dict_to_check
{"default_megacomplex": "decay"}
```

deprecate_module_attribute

glotaran.deprecation.deprecation_utils.**deprecate_module_attribute**(*, *deprecated_qual_name: str*, *new_qual_name: str*, *to_be_removed_in_version: str*, *module_load_overwrite: str = ''*) → Any

Import and return and anttribute from the new location.

This needs to be wrapped in the definition of a module wide `__getattr__` function so it won't throw warnings all the time (see example).

Parameters

- **deprecated_qual_name** (*str*) – Fully qualified name of the deprecated attribute e.g.: `glotaran.ParameterGroup`
- **new_qual_name** (*str*) – Fully qualified name of the new attribute e.g.: `glotaran.parameter.ParameterGroup`

- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **module_load_overwrite** (*str*) – Overwrite the location the functionality will be set from. This allows preserving functionality without polluting a new module with code just for the sake of it. By default ‘’

Returns
Module attribute from its new location.

Return type
Any

Raises
OverDueDeprecation – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *warn_deprecated*, *deprecate_submodule*

Examples

When deprecating the usage of `ParameterGroup` the root of `glotaran` and promoting to import it from `glotaran.parameter` the following code was added to the root `__init__.py`.

Listing 2: glotaran/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "ParameterGroup":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.ParameterGroup",
            new_qual_name="glotaran.parameter.ParameterGroup",
            to_be_removed_in_version="0.6.0",
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_
↪name}")
```

deprecate_submodule

glotaran.deprecation.deprecation_utils.**deprecate_submodule**(*, *deprecated_module_name: str*, *new_module_name: str*, *to_be_removed_in_version: str*, *module_load_overwrite: str = ''*) → module

Create a module at runtime which retrieves attributes from new module.

When moving a module, create a variable with the modules name in the parent packages `__init__.py`, so imports will be redirected to the new module location and a deprecation warning will be given,

to help the user adjust the outdated code. Each time an attribute is retrieved there will be a deprecation warning.

Parameters

- **deprecated_module_name** (*str*) – Fully qualified name of the deprecated module e.g.: `'glotaran.analysis.result'`
- **new_module_name** (*str*) – Fully qualified name of the new module e.g.: `'glotaran.project.result'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **module_load_overwrite** (*str*) – Overwrite the location for the new module the deprecated functionality is loaded from. This allows preserving functionality without polluting a new module with code just for the sake of it. By default ''

Returns
Module containing

Return type
ModuleType

Raises
OverDueDeprecation – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *deprecate_module_attribute*

Examples

When moving the module `result` from `glotaran.analysis.result` to `glotaran.project.result` the following code was added to the old parent packages (`glotaran.analysis`) `__init__.py`.

Listing 3: glotaran/analysis/__init__.py

```
from glotaran.deprecation.deprecation_utils import deprecate_submodule

result = deprecate_submodule(
    deprecated_module_name="glotaran.analysis.result",
    new_module_name="glotaran.project.result",
    to_be_removed_in_version="0.6.0",
)
```

glotaran_version

glotaran.deprecation.deprecation_utils.**glotaran_version**() → str

Version of the distribution.

This is basically the same as `glotaran.__version__` but independent from glotaran. This way all of the deprecation functionality can be used even in `glotaran.__init__.py` without moving the import below the definition of `__version__` or causeing a circular import issue.

Returns
The version string.

Return type
str

module_attribute

glotaran.deprecation.deprecation_utils.**module_attribute**(*module_qual_name: str*, *attribute_name: str*) → Any

Import and return the attribute (e.g. function or class) of a module.

This is basically the same as `from module_name import attribute_name as return_value` where this function returns `return_value`.

Parameters

- **module_qual_name** (*str*) – Fully qualified name for a module e.g. `glotaran.model.base_model`
- **attribute_name** (*str*) – Name of the attribute e.g. `Model`

Returns
Attribute of the module, e.g. a function or class.

Return type
Any

parse_version

glotaran.deprecation.deprecation_utils.**parse_version**(*version_str: str*) → tuple[int, int, int]

Parse version string to tuple of three ints for comparison.

Parameters
version_str (*str*) – Fully qualified version string of the form 'major.minor.patch'.

Returns
Version as tuple.

Return type
tuple[int, int, int]

Raises

- **ValueError** – If `version_str` has less that three elements separated by `.`.
- **ValueError** – If `version_str` 's first three elements can not be casted to int.

raise_deprecation_error

glotaran.deprecation.deprecation_utils.**raise_deprecation_error**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*) → NoReturn

Raise `GlotaranDeprectedApiError` error, with formatted message.

This should only be used if there is no reasonable way to keep the deprecated usage functional!

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Raises

- **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.
- **GlotaranDeprecatedApiError** – If `OverDueDeprecation` wasn't raised before.

warn_deprecated

glotaran.deprecation.deprecation_utils.**warn_deprecated**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*, *check_qual_names: tuple[bool, bool] = (True, True)*, *stacklevel: int = 2*, *importable_indices: tuple[int, int] = (1, 1)*) → None

Raise deprecation warning with change information.

The change information are old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **check_qual_names** (*tuple[bool, bool]*) – Whether or not to check for the existence `deprecated_qual_name_usage` and `deprecated_qual_name_usage`
  - Set the first value to False to prevent infinite recursion error when changing a module attribute import.
  - Set the second value to False if the new usage in in a different package or there is none.
- **stacklevel** (*int*) – Stack at which the warning should be shown as raise. Default: 2
- **importable_indices** (*tuple[int, int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use `importable_indices=(2, 1)`, this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`.

Raises

OverDueDeprecation – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` could deprecated and the usage of `load_model` being promoted instead.

Listing 4: glotaran/deprecation/modules/glotaran_root.py

```
def read_parameters_from_yaml_file(model_path: str):
    warn_deprecated(
        deprecated_qual_name_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
        new_qual_name_usage="glotaran.io.load_model.load_model(model_path)
↪",
        to_be_removed_in_version="0.6.0",
    )
    return load_model(model_path)
```

Exceptions

Exception Summary

`GlotaranApiDeprecationWarning`	Warning to give users about API changes.
`GlotaranDeprecatedApiError`	Exception raised when a deprecation has no replacement.
`OverDueDeprecation`	Error thrown when a deprecation should have been removed.

GlotaranApiDeprecationWarning

exception glotaran.deprecation.deprecation_utils.**GlotaranApiDeprecationWarning**

Warning to give users about API changes.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

GlotaranDeprecatedApiError

exception glotaran.deprecation.deprecation_utils.**GlotaranDeprecatedApiError**

Exception raised when a deprecation has no replacement.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

OverDueDeprecation

exception glotaran.deprecation.deprecation_utils.**OverDueDeprecation**

Error thrown when a deprecation should have been removed.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

modules

Package containing deprecated implementations which were removed.

To keep things organized the filenames should be like the relative import path from glotaran root, but with _ instead of `.`. E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

Modules

glotaran.deprecation.modules.builtin_io_yml	Deprecation functions for the yaml parser.
glotaran.deprecation.modules.examples	Deprecation package for 'glotaran.examples'.

builtin_io_yml

Deprecation functions for the yaml parser.

Functions

Summary

model_spec_deprecations	Check deprecations in the model specification `spec` dict.

model_spec_deprecations

glotaran.deprecation.modules.builtin_io_yml.**model_spec_deprecations**(*spec: MutableMapping[Any, Any]*) → None

Check deprecations in the model specification `spec` dict.

Parameters
spec (*MutableMapping[Any, Any]*) – Model specification dictionary

examples

Deprecation package for 'glotaran.examples'.

Modules

glotaran.deprecation.modules.examples.sequential	Deprecated functionality export for 'glotaran.examples.sequential'.

sequential

Deprecated functionality export for 'glotaran.examples.sequential'.

16.1.5 io

Functions for data IO.

Note:

Since Io functionality is purely plugin based this package mostly reexports functions from the `glotaran.plugin_system` from a common place.

Modules

glotaran.io.interface	Baseclasses to create Data/Project IO plugins from.
glotaran.io.prepare_dataset	Module containing dataset preparation functionality.
glotaran.io.preprocessor	Tools for data pre-processing.

interface

Baseclasses to create Data/Project IO plugins from.

The main purpose of those classes are to guarantee a consistent API via typechecker like `mypy` and demonstarate with methods are accessed by highlevel convenience functions for a given type of plugin.

To add additional options to a method, those options need to be keyword only arguments. See: https://www.python.org/dev/peps/pep-3102/

Classes

Summary

DataIoInterface	Baseclass for Data IO plugins.
ProjectIoInterface	Baseclass for Project IO plugins.
SavingOptions	A collection of options for result saving.

DataIoInterface

class glotaran.io.interface.**DataIoInterface**(*format_name: str*)

Bases: object

Baseclass for Data IO plugins.

Initialize a Data IO plugin with the name of the format.

Parameters
format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray.
save_dataset	Save data from xarray.Dataset to a file.

load_dataset

DataIoInterface.**load_dataset**(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray.

NOT IMPLEMENTED
Parameters
file_name (*str*) – File containing the data.
Returns
Data loaded from the file.
Return type
xr.Dataset|xr.DataArray

save_dataset

DataIoInterface.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED
Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray.

NOT IMPLEMENTED

Parameters

file_name (*str*) – File containing the data.

Returns

Data loaded from the file.

Return type

xr.Dataset|xr.DataArray

save_dataset(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file.

NOT IMPLEMENTED

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

ProjectIoInterface

class glotaran.io.interface.**ProjectIoInterface**(*format_name: str*)

Bases: object

Baseclass for Project IO plugins.

Initialize a Project IO plugin with the name of the format.

Parameters

format_name (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a Parameters instance from the specs defined in a file.
load_result	Create a Result instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file.
save_model	Save a Model instance to a spec file.
save_parameters	Save a Parameters instance to a spec file.
save_result	Save a Result instance to a spec file.
save_scheme	Save a Scheme instance to a spec file.

load_model

ProjectIoInterface.**load_model**(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the model specs.

Returns
Model instance created from the file.

Return type
Model

load_parameters

ProjectIoInterface.**load_parameters**(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
Parameters instance created from the file.

Return type
Parameters

load_result

ProjectIoInterface.**load_result**(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
result_path (*str*) – Path containing the result data.

Returns
Result instance created from the file.

Return type
Result

load_scheme

ProjectIoInterface.**load_scheme**(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

ProjectIoInterface.**save_model**(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

ProjectIoInterface.**save_parameters**(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters

- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

ProjectIoInterface.**save_result**(*result:* Result, *result_path: str*, *, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme

ProjectIoInterface.**save_scheme**(*scheme:* Scheme, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *Model*

Create a Model instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the model specs.

Returns
Model instance created from the file.

Return type
Model

load_parameters(*file_name: str*) → *Parameters*

Create a Parameters instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
Parameters instance created from the file.

Return type
Parameters

load_result(*result_path: str*) → *Result*

Create a Result instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
result_path (*str*) – Path containing the result data.

Returns
Result instance created from the file.

Return type
Result

load_scheme(*file_name: str*) → *Scheme*

Create a Scheme instance from the specs defined in a file.

NOT IMPLEMENTED

Parameters
file_name (*str*) – File containing the parameter specs.

Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* Model, *file_name: str*)

Save a Model instance to a spec file.

NOT IMPLEMENTED

Parameters
- **model** (*Model*) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* Parameters, *file_name: str*)

Save a Parameters instance to a spec file.

NOT IMPLEMENTED

Parameters
- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*, ***, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save a Result instance to a spec file.

NOT IMPLEMENTED

Parameters
- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.
- **saving_options** (SavingOptions) – Options for the saved result.

save_scheme(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file.

NOT IMPLEMENTED

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

SavingOptions

class glotaran.io.interface.**SavingOptions**(*data_filter: list[str] | None = None*, *data_format: Literal['nc'] = 'nc'*, *parameter_format: Literal['csv'] = 'csv'*, *report: bool = True*)

Bases: object

A collection of options for result saving.

Attributes Summary

data_filter
data_format
parameter_format
report

data_filter

SavingOptions.**data_filter: list[str] | None = None**

data_format

SavingOptions.**data_format: Literal['nc'] = 'nc'**

parameter_format

SavingOptions.**parameter_format: Literal['csv'] = 'csv'**

report

SavingOptions.**report: bool = True**

Methods Summary

Methods Documentation

data_filter: list[str] | None = None

data_format: Literal['nc'] = 'nc'

parameter_format: Literal['csv'] = 'csv'

report: bool = True

prepare_dataset

Module containing dataset preparation functionality.

Functions

Summary

add_svd_to_dataset	Add the SVD of a dataset inplace as Data variables to the dataset.
prepare_time_trace_dataset	Prepare a time trace `dataset` for global analysis.

add_svd_to_dataset

glotaran.io.prepare_dataset.**add_svd_to_dataset**(*dataset: xr.Dataset*, *name: str = 'data'*, *lsv_dim: Hashable = 'time'*, *rsv_dim: Hashable = 'spectral'*, *data_array: xr.DataArray | None = None*) → None

Add the SVD of a dataset inplace as Data variables to the dataset.

The SVD is only computed if it doesn't already exist on the dataset.

Parameters

- **dataset** (*xr.Dataset*) – Dataset the SVD values should be added to.
- **name** (*str*) – Key to access the datarray inside of the dataset, by default "data"
- **lsv_dim** (*Hashable*) – Name of the dimension for the left singular value, by default "time"
- **rsv_dim** (*Hashable*) – Name of the dimension for the right singular value, by default "spectral"
- **data_array** (*xr.DataArray | None*) – Dataarray to calculate the SVD for, when provided the data extraction from the dataset will be skipped, by default None

prepare_time_trace_dataset

glotaran.io.prepare_dataset.**prepare_time_trace_dataset**(*dataset: DataArray | Dataset*, *weight: ndarray | None = None*, *irf: ndarray | DataArray | None = None*) → Dataset

Prepare a time trace `dataset` for global analysis.

Parameters

- **dataset** (*xr.DataArray | xr.Dataset*) – Dataset to add data to.
- **weight** (*np.ndarray | None*) – A weight for the dataset.
- **irf** (*np.ndarray | xr.DataArray | None*) – An IRF for the dataset.

Returns
Original dataset with added SVD data and weight and/or irf data variables if provided.

Return type
xr.Dataset

Raises
ValueError – If an IRF with more than one dimensions was provided that isn't a xarray.DataArray.

preprocessor

Tools for data pre-processing.

Modules

glotaran.io.preprocessor.pipeline	A pre-processor pipeline for data.
glotaran.io.preprocessor.preprocessor	A pre-processor pipeline for data.

pipeline

A pre-processor pipeline for data.

Classes

Summary

PreProcessingPipeline	A pipeline for pre-processors.

PreProcessingPipeline

class glotaran.io.preprocessor.pipeline.**PreProcessingPipeline**(*, *actions: list[typing.Annotated[glotaran.io.preprocessor.p | glotaran.io.preprocessor.preprocessor.CorrectB FieldInfo(annotation=NoneType, required=True, discriminator='action')]] = None*)

Bases: `BaseModel`

A pipeline for pre-processors.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

Attributes Summary

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
actions	

model_computed_fields

PreProcessingPipeline.**model_computed_fields**

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config

PreProcessingPipeline.**model_config: ClassVar[ConfigDict] = {}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_extra

PreProcessingPipeline.**model_extra**

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields

PreProcessingPipeline.**model_fields: ClassVar[dict[str, FieldInfo]] = {'actions': FieldInfo(annotation=list[Annotated[Union[CorrectBaselineValue, CorrectBaselineAverage], FieldInfo(annotation=NoneType, required=True, discriminator='action')]], required=False, default_factory=list)}**

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_fields_set

PreProcessingPipeline.**model_fields_set**

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

actions

PreProcessingPipeline.**actions**: list[PipelineAction]

Methods Summary

apply	Apply all pre-processors on data.
construct	
copy	Returns a copy of the model.
correct_baseline_average	Correct a dataset by subtracting the average over a part of the data.
correct_baseline_value	Correct a dataset by subtracting baseline value.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

apply

PreProcessingPipeline.**apply**(*original: DataArray*) → DataArray

Apply all pre-processors on data.

Parameters

original (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

construct

classmethod PreProcessingPipeline.**construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy

PreProcessingPipeline.**copy**(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

correct_baseline_average

PreProcessingPipeline.**correct_baseline_average**(*select: dict[str, slice | list[int] | int] | None = None*, *exclude: dict[str, slice | list[int] | int] | None = None*) → *PreProcessingPipeline*

Correct a dataset by subtracting the average over a part of the data.

Parameters

- **select** (*dict[str, slice | list[int] | int] | None*) – The selection to average as dictionary of dimension and indexer. The indexer can be a slice, a list or an integer value.

- **exclude** (*dict[str, slice | list[int] | int] | None*) – Excluded regions from the average as dictionary of dimension and indexer. The indexer can be a slice, a list or an integer value.

Return type
PreProcessingPipeline

correct_baseline_value

PreProcessingPipeline.**correct_baseline_value**(*value: float*) → *PreProcessingPipeline*

Correct a dataset by subtracting baseline value.

Parameters
value (*float*) – The value to subtract.

Return type
PreProcessingPipeline

dict

PreProcessingPipeline.**dict**(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

from_orm

classmethod PreProcessingPipeline.**from_orm**(*obj: Any*) → *Model*

json

PreProcessingPipeline.**json**(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

model_construct

classmethod PreProcessingPipeline.**model_construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy

PreProcessingPipeline.**model_copy**(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump

PreProcessingPipeline.**model_dump**(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json

PreProcessingPipeline.**model_dump_json**(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of

strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

model_json_schema

classmethod PreProcessingPipeline.**model_json_schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

model_parametrized_name

classmethod PreProcessingPipeline.**model_parametrized_name**(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init

PreProcessingPipeline.**model_post_init**(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

model_rebuild

classmethod PreProcessingPipeline.**model_rebuild**(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

model_validate

classmethod PreProcessingPipeline.**model_validate**(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

model_validate_json

classmethod PreProcessingPipeline.**model_validate_json**(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

parse_file

classmethod PreProcessingPipeline.**parse_file**(*path: str | Path, *, content_type: str | None = None, encoding: str = 'utf8', proto: _deprecated_parse.Protocol | None = None, allow_pickle: bool = False*) → *Model*

parse_obj

classmethod PreProcessingPipeline.**parse_obj**(*obj: Any*) → *Model*

parse_raw

classmethod PreProcessingPipeline.**parse_raw**(*b: str | bytes, *, content_type: str | None = None, encoding: str = 'utf8', proto: _deprecated_parse.Protocol | None = None, allow_pickle: bool = False*) → *Model*

schema

classmethod PreProcessingPipeline.**schema**(*by_alias: bool = True, ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

schema_json

classmethod PreProcessingPipeline.**schema_json**(*, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

update_forward_refs

classmethod PreProcessingPipeline.**update_forward_refs**(***localns: Any*) → None

validate

classmethod PreProcessingPipeline.**validate**(*value: Any*) → *Model*

Methods Documentation

actions: list[PipelineAction]

apply(*original: DataArray*) → DataArray

Apply all pre-processors on data.

Parameters

original (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

classmethod construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

correct_baseline_average(*select: dict[str, slice | list[int] | int] | None = None*, *exclude: dict[str, slice | list[int] | int] | None = None*) → *PreProcessingPipeline*

Correct a dataset by subtracting the average over a part of the data.

Parameters

- **select** (*dict[str, slice | list[int] | int] | None*) – The selection to average as dictionary of dimension and indexer. The indexer can be a slice, a list or an integer value.
- **exclude** (*dict[str, slice | list[int] | int] | None*) – Excluded regions from the average as dictionary of dimension and indexer. The indexer can be a slice, a list or an integer value.

Return type

PreProcessingPipeline

correct_baseline_value(*value: float*) → *PreProcessingPipeline*

Correct a dataset by subtracting baseline value.

Parameters

value (*float*) – The value to subtract.

Return type

PreProcessingPipeline

dict(**, include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False*) → Dict[str, Any]

classmethod from_orm(*obj: Any*) → *Model*

json(**, include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False, encoder: Callable[[Any], Any] | None = PydanticUndefined, models_as_dict: bool = PydanticUndefined, **dumps_kwargs: Any*) → str

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(**, update: dict[str, Any] | None = None, deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.
deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:
mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.
include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'actions': FieldInfo(annotation=list[Annotated[Union[CorrectBaselineValue, CorrectBaselineAverage], FieldInfo(annotation=NoneType, required=True, discriminator='action')]], required=False, default_factory=list)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod parse_file(*path: str | Path*, *, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod parse_obj(*obj: Any*) → *Model*

classmethod parse_raw(*b: str | bytes*, *, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

classmethod schema_json(*, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

classmethod update_forward_refs(***localns: Any*) → None

classmethod validate(*value: Any*) → *Model*

preprocessor

A pre-processor pipeline for data.

Classes

Summary

CorrectBaselineAverage	Corrects a dataset by subtracting the average over a part of the data.
CorrectBaselineValue	Corrects a dataset by subtracting baseline value.
PreProcessor	A base class for pre=processors.

CorrectBaselineAverage

class glotaran.io.preprocessor.preprocessor.**CorrectBaselineAverage**(*, *action: Literal['baseline-average'] = 'baseline-average'*, *select: dict[str, slice | list[int] | int] | None = None*, *exclude: dict[str, slice | list[int] | int] | None = None*)

Bases: *PreProcessor*

Corrects a dataset by subtracting the average over a part of the data.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

Attributes Summary

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
action	
select	
exclude	

model_computed_fields

CorrectBaselineAverage.**model_computed_fields**

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config

CorrectBaselineAverage.**model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_extra

CorrectBaselineAverage.**model_extra**

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields

CorrectBaselineAverage.**model_fields: ClassVar[dict[str, FieldInfo]] = {'action': FieldInfo(annotation=Literal[str], required=False, default='baseline-average'), 'exclude': FieldInfo(annotation=Union[dict[str, Union[slice, list[int], int]], NoneType], required=False), 'select': FieldInfo(annotation=Union[dict[str, Union[slice, list[int], int]], NoneType], required=False)}**

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_fields_set

CorrectBaselineAverage.**model_fields_set**

Returns the set of fields that have been set on this model instance.
Returns:
A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

action

CorrectBaselineAverage.**action: Literal['baseline-average']**

select

CorrectBaselineAverage.**select: dict[str, slice | list[int] | int] | None**

exclude

CorrectBaselineAverage.**exclude: dict[str, slice | list[int] | int] | None**

Methods Summary

apply	Apply the pre-processor.
construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

apply

CorrectBaselineAverage.**apply**(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

construct

classmethod CorrectBaselineAverage.**construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy

CorrectBaselineAverage.**copy**(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

dict

CorrectBaselineAverage.**dict**(***, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

from_orm

classmethod CorrectBaselineAverage.**from_orm**(*obj: Any*) → *Model*

json

CorrectBaselineAverage.**json**(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

model_construct

classmethod CorrectBaselineAverage.**model_construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy

CorrectBaselineAverage.**model_copy**(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump

CorrectBaselineAverage.**model_dump**(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.

If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json

CorrectBaselineAverage.**model_dump_json**(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

model_json_schema

classmethod CorrectBaselineAverage.**model_json_schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

model_parametrized_name

classmethod CorrectBaselineAverage.**model_parametrized_name**(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init

CorrectBaselineAverage.**model_post_init**(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

model_rebuild

classmethod CorrectBaselineAverage.**model_rebuild**(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

model_validate

classmethod CorrectBaselineAverage.**model_validate**(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

model_validate_json

classmethod CorrectBaselineAverage.**model_validate_json**(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

parse_file

classmethod CorrectBaselineAverage.**parse_file**(*path: str | Path*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

parse_obj

classmethod CorrectBaselineAverage.**parse_obj**(*obj: Any*) → *Model*

parse_raw

classmethod CorrectBaselineAverage.**parse_raw**(*b: str | bytes*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

schema

classmethod CorrectBaselineAverage.**schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

schema_json

classmethod CorrectBaselineAverage.**schema_json**(***, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

update_forward_refs

classmethod CorrectBaselineAverage.**update_forward_refs**(***localns: Any*) → None

validate

classmethod CorrectBaselineAverage.**validate**(*value: Any*) → *Model*

Methods Documentation

class Config

Bases: object

Config for BaseModel.

arbitrary_types_allowed = True

action: Literal['baseline-average']

apply(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

classmethod construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

dict(***, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

exclude: dict[str, slice | list[int] | int] | None

classmethod from_orm(*obj: Any*) → *Model*

json(***, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(***, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'action': FieldInfo(annotation=Literal[str], required=False, default='baseline-average'), 'exclude': FieldInfo(annotation=Union[dict[str, Union[slice, list[int], int]], NoneType], required=False), 'select': FieldInfo(annotation=Union[dict[str, Union[slice, list[int], int]], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of
GenerateJsonSchema with your desired modifications
mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod parse_file(*path: str | Path*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod parse_obj(*obj: Any*) → *Model*

classmethod parse_raw(*b: str | bytes*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

classmethod schema_json(***, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

select: dict[str, slice | list[int] | int] | None

classmethod update_forward_refs(***localns: Any*) → None

classmethod validate(*value: Any*) → *Model*

CorrectBaselineValue

class glotaran.io.preprocessor.preprocessor.**CorrectBaselineValue**(***, *action: Literal['baseline-value'] = 'baseline-value'*, *value: float*)

Bases: *PreProcessor*

Corrects a dataset by subtracting baseline value.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

Attributes Summary

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
action	
value	

model_computed_fields

CorrectBaselineValue.**model_computed_fields**

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config

CorrectBaselineValue.**model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_extra

CorrectBaselineValue.**model_extra**

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields

CorrectBaselineValue.**model_fields: ClassVar[dict[str, FieldInfo]] = {'action': FieldInfo(annotation=Literal[str], required=False, default='baseline-value'), 'value': FieldInfo(annotation=float, required=True)}**

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_fields_set

CorrectBaselineValue.**model_fields_set**

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

action

CorrectBaselineValue.**action: Literal['baseline-value']**

value

CorrectBaselineValue.**value:** float

Methods Summary

apply	Apply the pre-processor.
construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

apply

CorrectBaselineValue.**apply**(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

construct

classmethod CorrectBaselineValue.**construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy

CorrectBaselineValue.**copy**(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

dict

CorrectBaselineValue.**dict**(***, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

from_orm

classmethod CorrectBaselineValue.**from_orm**(*obj: Any*) → *Model*

json

CorrectBaselineValue.**json**(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

model_construct

classmethod CorrectBaselineValue.**model_construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy

CorrectBaselineValue.**model_copy**(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump

CorrectBaselineValue.**model_dump**(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.

If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json

CorrectBaselineValue.**model_dump_json**(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

model_json_schema

classmethod CorrectBaselineValue.**model_json_schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

model_parametrized_name

classmethod CorrectBaselineValue.**model_parametrized_name**(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init

CorrectBaselineValue.**model_post_init**(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

model_rebuild

classmethod CorrectBaselineValue.**model_rebuild**(*, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

model_validate

classmethod CorrectBaselineValue.**model_validate**(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

model_validate_json

classmethod CorrectBaselineValue.**model_validate_json**(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

parse_file

classmethod CorrectBaselineValue.**parse_file**(*path: str | Path*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

parse_obj

classmethod CorrectBaselineValue.**parse_obj**(*obj: Any*) → *Model*

parse_raw

classmethod CorrectBaselineValue.**parse_raw**(*b: str | bytes*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

schema

classmethod CorrectBaselineValue.**schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

schema_json

classmethod CorrectBaselineValue.**schema_json**(***, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

update_forward_refs

classmethod CorrectBaselineValue.**update_forward_refs**(***localns: Any*) → None

validate

classmethod CorrectBaselineValue.**validate**(*value: Any*) → *Model*

Methods Documentation

class Config

Bases: `object`

Config for BaseModel.

arbitrary_types_allowed = True

action: Literal['baseline-value']

apply(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Return type

xr.DataArray

classmethod construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self. model_validate(data) `

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

dict(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

classmethod from_orm(*obj: Any*) → *Model*

json(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'action': FieldInfo(annotation=Literal[str], required=False, default='baseline-value'), 'value': FieldInfo(annotation=float, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.
Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.
Raises:
ValidationError: If the object could not be validated.
Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.
Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.
Returns:
The validated Pydantic model.
Raises:
ValueError: If *json_data* is not a JSON string.

classmethod parse_file(*path: str | Path*, *, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod parse_obj(*obj: Any*) → *Model*

classmethod parse_raw(*b: str | bytes*, *, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

classmethod schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

classmethod schema_json(*, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

classmethod update_forward_refs(***localns: Any*) → None

classmethod validate(*value: Any*) → *Model*

value: float

PreProcessor

class glotaran.io.preprocessor.preprocessor.**PreProcessor**

Bases: BaseModel, ABC

A base class for pre=processors.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

Attributes Summary

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.

model_computed_fields

PreProcessor.**model_computed_fields**

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config

PreProcessor.**model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_extra

PreProcessor.**model_extra**

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields

PreProcessor.**model_fields: ClassVar[dict[str, FieldInfo]] = {}**

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

model_fields_set

PreProcessor.**model_fields_set**

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,

i.e. that were not filled from defaults.

Methods Summary

apply	Apply the pre-processor.
construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

apply

abstract PreProcessor.**apply**(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Returns

- *xr.DataArray*
- **.. # noqa** (*DAR202*)

construct

classmethod PreProcessor.**construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy

PreProcessor.**copy**(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude,
round_trip=True) data = {**data, **(update or {})} copied = self.
model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping

specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values

in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

dict

PreProcessor.**dict**(***, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

from_orm

classmethod PreProcessor.**from_orm**(*obj: Any*) → *Model*

json

PreProcessor.**json**(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

model_construct

classmethod PreProcessor.**model_construct**(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy

PreProcessor.**model_copy**(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump

PreProcessor.**model_dump**(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.

If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json

PreProcessor.**model_dump_json**(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

model_json_schema

classmethod PreProcessor.**model_json_schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

model_parametrized_name

classmethod PreProcessor.**model_parametrized_name**(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init

PreProcessor.**model_post_init**(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

model_rebuild

classmethod PreProcessor.**model_rebuild**(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

model_validate

classmethod PreProcessor.**model_validate**(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

model_validate_json

classmethod PreProcessor.**model_validate_json**(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

parse_file

classmethod PreProcessor.**parse_file**(*path: str | Path*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

parse_obj

classmethod PreProcessor.**parse_obj**(*obj: Any*) → *Model*

parse_raw

classmethod PreProcessor.**parse_raw**(*b: str | bytes*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → *Model*

schema

classmethod PreProcessor.**schema**(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

schema_json

classmethod PreProcessor.**schema_json**(***, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

update_forward_refs

classmethod PreProcessor.**update_forward_refs**(***localns: Any*) → None

validate

classmethod PreProcessor.**validate**(*value: Any*) → *Model*

Methods Documentation

class Config

Bases: object

Config for BaseModel.

arbitrary_types_allowed = True

abstract apply(*data: DataArray*) → DataArray

Apply the pre-processor.

Parameters

data (*xr.DataArray*) – The data to process.

Returns

- *xr.DataArray*
- **.. # noqa** (*DAR202*)

classmethod construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

copy(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.
If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self. model_validate(data) `

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

dict(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

classmethod from_orm(*obj: Any*) → *Model*

json(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → *Model*

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → *Model*

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of
GenerateJsonSchema with your desired modifications
mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → *Model*

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.
Raises:
ValidationError: If the object could not be validated.
Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → *Model*

Validate the given JSON data against the Pydantic model.
Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.
Returns:
The validated Pydantic model.
Raises:
ValueError: If *json_data* is not a JSON string.

classmethod parse_file(*path: str | Path, *, content_type: str | None = None, encoding: str = 'utf8', proto: _deprecated_parse.Protocol | None = None, allow_pickle: bool = False*) → *Model*

classmethod parse_obj(*obj: Any*) → *Model*

classmethod parse_raw(*b: str | bytes, *, content_type: str | None = None, encoding: str = 'utf8', proto: _deprecated_parse.Protocol | None = None, allow_pickle: bool = False*) → *Model*

classmethod schema(*by_alias: bool = True, ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

classmethod schema_json(**, by_alias: bool = True, ref_template: str = '#/$defs/{model}', **dumps_kwargs: Any*) → str

classmethod update_forward_refs(***localns: Any*) → None

classmethod validate(*value: Any*) → *Model*

16.1.6 model

The glotaran model package.

Modules

glotaran.model.clp_constraint	This module contains clp constraint items.
glotaran.model.clp_penalties	This module contains clp penalty items.
glotaran.model.clp_relation	This module contains clp relation items.
glotaran.model.dataset_group	This module contains the dataset group.
glotaran.model.dataset_model	This module contains the dataset model.
glotaran.model.interval_item	This module contains the interval item.
glotaran.model.item(cls)	Create an item from a class.
glotaran.model.megacomplex(*[, ...])	Create a megacomplex from a class.
glotaran.model.model	This module contains the model.
glotaran.model.weight	This module contains weight item.

clp_constraint

This module contains clp constraint items.

Classes

Summary

ClpConstraint	Baseclass for clp constraints.
OnlyConstraint	Constraints the target to 0 outside the given interval.
ZeroConstraint	Constraints the target to 0 in the given interval.

ClpConstraint

class glotaran.model.clp_constraint.**ClpConstraint**(*, *interval: tuple[float, float] | list[tuple[float, float]] | None = None*, *type: str*, *target: str*)

Bases: TypedItem, IntervalItem

Baseclass for clp constraints.

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

Method generated by attrs for class ClpConstraint.

Attributes Summary

target
type
interval

target

ClpConstraint.**target**: str

type

ClpConstraint.**type**: str

interval

ClpConstraint.**interval**: tuple[float, float] | list[tuple[float, float]] | None

Methods Summary

applies	Check if the index is in the intervals.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
has_interval	Check if intervals are defined.

applies

ClpConstraint.**applies**(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters

index (*float*) – The index.

Return type

bool

get_item_type

classmethod ClpConstraint.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod ClpConstraint.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod ClpConstraint.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

has_interval

ClpConstraint.**has_interval**() → bool

Check if intervals are defined.

Return type
bool

Methods Documentation

applies(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

has_interval() → bool

Check if intervals are defined.

Return type
bool

interval: tuple[float, float] | list[tuple[float, float]] | None

target: str

type: str

OnlyConstraint

class glotaran.model.clp_constraint.**OnlyConstraint**(*, *interval: tuple[float, float] | list[tuple[float, float]] | None = None*, *target: str*, *type: str = 'only'*)

Bases: *ZeroConstraint*

Constraints the target to 0 outside the given interval.

Method generated by attrs for class OnlyConstraint.

Attributes Summary

type
target
interval

type

OnlyConstraint.**type: str**

target

OnlyConstraint.**target**: str

interval

OnlyConstraint.**interval**: tuple[float, float] | list[tuple[float, float]] | None

Methods Summary

applies	Check if the constraint applies on this index.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
has_interval	Check if intervals are defined.

applies

OnlyConstraint.**applies**(*index: float | None*) → bool

Check if the constraint applies on this index.

Parameters
index (*float*) – The index.

Return type
bool

get_item_type

classmethod OnlyConstraint.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod OnlyConstraint.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod OnlyConstraint.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

has_interval

OnlyConstraint.**has_interval**() → bool

Check if intervals are defined.

Return type
bool

Methods Documentation

applies(*index: float | None*) → bool

Check if the constraint applies on this index.

Parameters
index (*float*) – The index.

Return type
bool

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

has_interval() → bool

Check if intervals are defined.

Return type
bool

interval: tuple[float, float] | list[tuple[float, float]] | None

target: str

type: str

ZeroConstraint

class glotaran.model.clp_constraint.**ZeroConstraint**(*, *interval: tuple[float, float] | list[tuple[float, float]] | None = None*, *target: str*, *type: str = 'zero'*)

Bases: *ClpConstraint*

Constraints the target to 0 in the given interval.

Method generated by attrs for class ZeroConstraint.

Attributes Summary

type
target
interval

type

ZeroConstraint.**type: str**

target

ZeroConstraint.**target: str**

interval

ZeroConstraint.**interval: tuple[float, float] | list[tuple[float, float]] | None**

Methods Summary

applies	Check if the index is in the intervals.
get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.
has_interval	Check if intervals are defined.

applies

ZeroConstraint.**applies**(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

get_item_type

classmethod ZeroConstraint.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod ZeroConstraint.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod ZeroConstraint.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

has_interval

ZeroConstraint.**has_interval**() → bool

Check if intervals are defined.

Return type
bool

Methods Documentation

applies(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

has_interval() → bool

Check if intervals are defined.

Return type
bool

interval: tuple[float, float] | list[tuple[float, float]] | None

target: str

type: str

clp_penalties

This module contains clp penalty items.

Classes

Summary

ClpPenalty	Baseclass for clp penalties.
EqualAreaPenalty	Forces the area of 2 clp to be the same.

ClpPenalty

class glotaran.model.clp_penalties.**ClpPenalty**(*, *type: str*)

Bases: TypedItem

Baseclass for clp penalties.

Method generated by attrs for class ClpPenalty.

Attributes Summary

type

type

ClpPenalty.**type: str**

Methods Summary

get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

get_item_type

classmethod ClpPenalty.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod ClpPenalty.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod ClpPenalty.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

type: str

EqualAreaPenalty

class glotaran.model.clp_penalties.**EqualAreaPenalty**(*, *type: str = 'equal_area'*, *source: str*, *source_intervals: list[tuple[float, float]]*, *target: str*, *target_intervals: list[tuple[float, float]]*, *parameter: Parameter | str*, *weight: float*)

Bases: *ClpPenalty*

Forces the area of 2 clp to be the same.

An equal area constraint adds a the difference of the sum of a compartments in the e matrix in one or 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.

Method generated by attrs for class EqualAreaPenalty.

Attributes Summary

type
source
source_intervals
target
target_intervals
parameter
weight

type

EqualAreaPenalty.**type**: str

source

EqualAreaPenalty.**source**: str

source_intervals

EqualAreaPenalty.**source_intervals**: list[tuple[float, float]]

target

EqualAreaPenalty.**target**: str

target_intervals

EqualAreaPenalty.**target_intervals**: list[tuple[float, float]]

parameter

EqualAreaPenalty.**parameter: ParameterType**

weight

EqualAreaPenalty.**weight: float**

Methods Summary

get_item_type	Get the type string.
get_item_type_class	Get the type for a type string.
get_item_types	Get all type strings.

get_item_type

classmethod EqualAreaPenalty.**get_item_type**() → str

Get the type string.

Return type
str

get_item_type_class

classmethod EqualAreaPenalty.**get_item_type_class**(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

get_item_types

classmethod EqualAreaPenalty.**get_item_types**() → list[str]

Get all type strings.

Return type
list[str]

Methods Documentation

classmethod get_item_type() → str

Get the type string.

Return type
str

classmethod get_item_type_class(*item_type: str*) → Type

Get the type for a type string.

Parameters
item_type (*str*) – The type string.

Return type
Type

classmethod get_item_types() → list[str]

Get all type strings.

Return type
list[str]

parameter: ParameterType

source: str

source_intervals: list[tuple[float, float]]

target: str

target_intervals: list[tuple[float, float]]

type: str

weight: float

clp_relation

This module contains clp relation items.

Classes

Summary

ClpRelation	Applies a relation between two clps.

ClpRelation

class glotaran.model.clp_relation.**ClpRelation**(*, *interval: tuple[float, float] | list[tuple[float, float]] | None = None*, *source: str*, *target: str*, *parameter: Parameter | str*)

Bases: *IntervalItem*

Applies a relation between two clps.

The relation is applied as $target = parameter * source$.

Method generated by attrs for class ClpRelation.

Attributes Summary

source
target
parameter
interval

source

ClpRelation.**source**: str

target

ClpRelation.**target**: str

parameter

ClpRelation.**parameter**: *Parameter* | str

interval

ClpRelation.**interval**: tuple[float, float] | list[tuple[float, float]] | None

Methods Summary

applies	Check if the index is in the intervals.
has_interval	Check if intervals are defined.

applies

ClpRelation.**applies**(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

has_interval

ClpRelation.**has_interval**() → bool

Check if intervals are defined.

Return type
bool

Methods Documentation

applies(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

has_interval() → bool

Check if intervals are defined.

Return type
bool

interval: tuple[float, float] | list[tuple[float, float]] | None

parameter: *Parameter* | str

source: str

target: str

dataset_group

This module contains the dataset group.

Classes

Summary

DatasetGroup	A dataset group for optimization.
DatasetGroupModel	A group of datasets which will evaluated independently.

DatasetGroup

class glotaran.model.dataset_group.**DatasetGroup**(*residual_function: Literal['variable_projection', 'non_negative_least_squares']*, *link_clp: bool | None*, *model:* Model, *parameters:* Parameters | *None* = *None*, *dataset_models: dict[str,* DatasetModel*]* = *_Nothing.NOTHING*)

Bases: object

A dataset group for optimization.

Method generated by attrs for class DatasetGroup.

Attributes Summary

residual_function	The residual function to use.
link_clp	Whether to link the clp parameter.
model	
parameters	
dataset_models	

residual_function

DatasetGroup.**residual_function: Literal['variable_projection', 'non_negative_least_squares']**

The residual function to use.

link_clp

DatasetGroup.**link_clp**: bool | None

Whether to link the clp parameter.

model

DatasetGroup.**model**: *Model*

parameters

DatasetGroup.**parameters**: *Parameters* | None

dataset_models

DatasetGroup.**dataset_models**: dict[str, *DatasetModel*]

Methods Summary

is_linkable	Check if the group is linkable.
set_parameters	Set the group parameters.

is_linkable

DatasetGroup.**is_linkable**(*parameters:* Parameters, *data: Mapping[str, xr.Dataset]*) → bool

Check if the group is linkable.

Parameters

- **parameters** (Parameters) – A parameter set parameters.
- **data** (*Mapping[str, xr.Dataset]*) – A the data to link.

Return type

bool

set_parameters

DatasetGroup.**set_parameters**(*parameters:* Parameters)

Set the group parameters.

Parameters

parameters (Parameters) – The parameters.

Methods Documentation

dataset_models: dict[str, *DatasetModel*]

is_linkable(*parameters:* Parameters, *data: Mapping[str, xr.Dataset]*) → bool

Check if the group is linkable.

Parameters

- **parameters** (Parameters) – A parameter set parameters.
- **data** (*Mapping[str, xr.Dataset]*) – A the data to link.

Return type

bool

link_clp: bool | None

Whether to link the clp parameter.

model: *Model*

parameters: *Parameters* | None

residual_function: Literal['variable_projection', 'non_negative_least_squares']

The residual function to use.

set_parameters(*parameters:* Parameters)

Set the group parameters.

Parameters

parameters (Parameters) – The parameters.

DatasetGroupModel

class glotaran.model.dataset_group.**DatasetGroupModel**(*, *label: str*, *residual_function: Literal['variable_projection', 'non_negative_least_squares'] = 'variable_projection'*, *link_clp: bool | None = None*)

Bases: ModelItem

A group of datasets which will evaluated independently.

Method generated by attrs for class DatasetGroupModel.

Attributes Summary

residual_function	The residual function to use.
link_clp	Whether to link the clp parameter.
label	

residual_function

DatasetGroupModel.**residual_function: Literal['variable_projection', 'non_negative_least_squares']**

The residual function to use.

link_clp

DatasetGroupModel.**link_clp: bool | None**

Whether to link the clp parameter.

label

DatasetGroupModel.**label: str**

Methods Summary

Methods Documentation

label: str

link_clp: bool | None

Whether to link the clp parameter.

residual_function: Literal['variable_projection', 'non_negative_least_squares']

The residual function to use.

dataset_model

This module contains the dataset model.

Functions

Summary

finalize_dataset_model	Finalize a dataset by applying all megacomplex finalize methods.
get_dataset_model_model_dimension	Get the dataset model's model dimension.
get_megacomplex_issues	Get issues for megacomplexes.
has_dataset_model_global_model	Check if the dataset model can model the global dimension.
iterate_dataset_model_global_megacomple	Iterate the dataset model's global megacomplexes.
iterate_dataset_model_megacomplexes	Iterate the dataset model's megacomplexes.
validate_global_megacomplexes	Get issues for dataset model global megacomplexes.
validate_megacomplexes	Get issues for dataset model megacomplexes.

finalize_dataset_model

glotaran.model.dataset_model.**finalize_dataset_model**(*dataset_model:* DatasetModel, *dataset: Dataset*)

Finalize a dataset by applying all megacomplex finalize methods.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **dataset** (*xr.Dataset*) – The dataset.

get_dataset_model_model_dimension

glotaran.model.dataset_model.**get_dataset_model_model_dimension**(*dataset_model:* DatasetModel) → str

Get the dataset model's model dimension.

Parameters
dataset_model (DatasetModel) – The dataset model.

Return type
str

Raises
ValueError – Raised if the dataset model does not have megacomplexes or if it is not filled.

get_megacomplex_issues

glotaran.model.dataset_model.**get_megacomplex_issues**(*value: list[str | Megacomplex] | None*, *model:* Model, *is_global: bool*) → list[ItemIssue]

Get issues for megacomplexes.

Parameters

- **value** (*list[str | Megacomplex] | None*) – A list of megacomplexes.

- **model** (Model) – The model.
- **is_global** (*bool*) – Whether the megacomplexes are global.

Return type
list[ItemIssue]

has_dataset_model_global_model

glotaran.model.dataset_model.**has_dataset_model_global_model**(*dataset_model:* DatasetModel) → bool

Check if the dataset model can model the global dimension.

Parameters
dataset_model (DatasetModel) – The dataset model.

Return type
bool

iterate_dataset_model_global_megacomplexes

glotaran.model.dataset_model.**iterate_dataset_model_global_megacomplexes**(*dataset_model:* DatasetModel) → Generator[tuple[*Parameter* | str | None, Megacomplex | str], None, None]

Iterate the dataset model's global megacomplexes.

Parameters
dataset_model (DatasetModel) – The dataset model.

Yields
tuple[Parameter | str | None, Megacomplex | str] – A scale and megacomplex.

iterate_dataset_model_megacomplexes

glotaran.model.dataset_model.**iterate_dataset_model_megacomplexes**(*dataset_model:* DatasetModel) → Generator[tuple[*Parameter* | str | None, Megacomplex | str], None, None]

Iterate the dataset model's megacomplexes.

Parameters
dataset_model (DatasetModel) – The dataset model.

Yields
tuple[Parameter | str | None, Megacomplex | str] – A scale and megacomplex.

validate_global_megacomplexes

glotaran.model.dataset_model.**validate_global_megacomplexes**(*value: list[str | Megacomplex] | None*, *dataset_model:* DatasetModel, *model:* Model, *parameters:* Parameters | *None*) → list[ItemIssue]

Get issues for dataset model global megacomplexes.

Parameters

- **value** (*list[str | Megacomplex] | None*) – A list of megacomplexes.
- **dataset_model** (DatasetModel) – The dataset model.
- **model** (Model) – The model.
- **parameters** (Parameters | *None*,) – The parameters.

Return type
list[ItemIssue]

validate_megacomplexes

glotaran.model.dataset_model.**validate_megacomplexes**(*value: list[str | Megacomplex]*, *dataset_model:* DatasetModel, *model:* Model, *parameters:* Parameters | *None*) → list[ItemIssue]

Get issues for dataset model megacomplexes.

Parameters

- **value** (*list[str | Megacomplex]*) – A list of megacomplexes.
- **dataset_model** (DatasetModel) – The dataset model.
- **model** (Model) – The model.
- **parameters** (Parameters | *None*,) – The parameters.

Return type
list[ItemIssue]

Classes

Summary

DatasetModel	A model for datasets.
ExclusiveMmegacomplexIssue	Issue for exclusive megacomplexes.
UniqueMegacomplexIssue	Issue for unique megacomplexes.

DatasetModel

class glotaran.model.dataset_model.**DatasetModel**(*, *label: str*, *group: str = 'default'*, *force_index_dependent: bool = False*, *megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]]*, *megacomplex_scale: list[glotaran.parameter.parameter.Parameter | str] | None = None*, *global_megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None = None*, *global_megacomplex_scale: list[glotaran.parameter.parameter.Parameter | str] | None = None*, *scale: Parameter | str | None = None*)

Bases: ModelItem

A model for datasets.

Method generated by attrs for class DatasetModel.

Attributes Summary

group
force_index_dependent
megacomplex
megacomplex_scale
global_megacomplex
global_megacomplex_scale
scale
label

group

DatasetModel.**group**: str

force_index_dependent

DatasetModel.**force_index_dependent**: bool

megacomplex

DatasetModel.**megacomplex**:
list[Union[glotaran.model.megacomplex.Megacomplex, str]]

megacomplex_scale

DatasetModel.**megacomplex_scale**: list[*glotaran.parameter.parameter.Parameter*
| str] | None

global_megacomplex

DatasetModel.**global_megacomplex**:
list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None

global_megacomplex_scale

DatasetModel.**global_megacomplex_scale**:
list[*glotaran.parameter.parameter.Parameter* | str] | None

scale

DatasetModel.**scale**: *Parameter* | str | None

label

DatasetModel.**label**: str

Methods Summary

Methods Documentation

force_index_dependent: bool

global_megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]] | None

global_megacomplex_scale: list[*glotaran.parameter.parameter.Parameter* | str] | None

group: str

label: str

megacomplex: list[Union[glotaran.model.megacomplex.Megacomplex, str]]

megacomplex_scale: list[*glotaran.parameter.parameter.Parameter* | str] | None

scale: *Parameter* | str | None

ExclusiveMegacomplexIssue

class glotaran.model.dataset_model.**ExclusiveMegacomplexIssue**(*label: str*, *megacomplex_type: str*, *is_global: bool*)

Bases: ItemIssue

Issue for exclusive megacomplexes.

Create an ExclusiveMegacomplexIssue.

Parameters

- **label** (*str*) – The megacomplex label.
- **megacomplex_type** (*str*) – The megacomplex type.
- **is_global** (*bool*) – Whether the megacomplex is global.

Methods Summary

to_string	Get the issue as string.

to_string

ExclusiveMegacomplexIssue.**to_string**() → str

Get the issue as string.

Return type

str

Methods Documentation

to_string() → str

Get the issue as string.

Return type

str

UniqueMegacomplexIssue

class glotaran.model.dataset_model.**UniqueMegacomplexIssue**(*label: str*, *megacomplex_type: str*, *is_global: bool*)

Bases: ItemIssue

Issue for unique megacomplexes.

Create a UniqueMegacomplexIssue.

Parameters

- **label** (*str*) – The megacomplex label.
- **megacomplex_type** (*str*) – The megacomplex type.
- **is_global** (*bool*) – Whether the megacomplex is global.

Methods Summary

to_string	Get the issue as string.

to_string

UniqueMegacomplexIssue.**to_string**()

Get the issue as string.

Return type

str

Methods Documentation

to_string()

Get the issue as string.

Return type

str

interval_item

This module contains the interval item.

Classes

Summary

IntervalItem	An item with an interval.

IntervalItem

class glotaran.model.interval_item.**IntervalItem**(*, *interval: tuple[float, float] | list[tuple[float, float]] | None = None*)

Bases: Item

An item with an interval.

Method generated by attrs for class IntervalItem.

Attributes Summary

interval

interval

IntervalItem.**interval: tuple[float, float] | list[tuple[float, float]] | None**

Methods Summary

applies	Check if the index is in the intervals.
has_interval	Check if intervals are defined.

applies

IntervalItem.**applies**(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

has_interval

IntervalItem.**has_interval**() → bool

Check if intervals are defined.

Return type
bool

Methods Documentation

applies(*index: float | None*) → bool

Check if the index is in the intervals.

Parameters
index (*float*) – The index.

Return type
bool

has_interval() → bool

Check if intervals are defined.

Return type
bool

interval: tuple[float, float] | list[tuple[float, float]] | None

item

glotaran.model.**item**(*cls: type[ItemT]*) → type[ItemT]

Create an item from a class.

Parameters
cls (*type[ItemT]*) – The class.

Return type
type[ItemT]

megacomplex

glotaran.model.**megacomplex**(*, *dataset_model_type: type[DatasetModel] | None = None*, *exclusive: bool = False*, *unique: bool = False*) → Callable

Create a megacomplex from a class.

Parameters

- **dataset_model_type** (*type*) – The dataset model type.
- **exclusive** (*bool*) – Whether the megacomplex is exclusive.
- **unique** (*bool*) – Whether the megacomplex is unique.

Return type
Callable

model

This module contains the model.

Classes

Summary

Model	A model for global target analysis.

Model

class glotaran.model.model.**Model**(*, *clp_penalties=_Nothing.NOTHING*, *clp_constraints=_Nothing.NOTHING*, *clp_relations=_Nothing.NOTHING*, *dataset_groups: dict[str, Union[glotaran.model.dataset_group.DatasetGroupModel, Any]] = _Nothing.NOTHING*, *dataset: dict[str, glotaran.model.dataset_model.DatasetModel]*, *megacomplex=_Nothing.NOTHING*, *weights=_Nothing.NOTHING*)

Bases: object

A model for global target analysis.

Method generated by attrs for class Model.

Attributes Summary

source_path
clp_penalties
clp_constraints
clp_relations
dataset_groups
dataset
megacomplex
weights

source_path

Model.**source_path**: str | None

clp_penalties

Model.**clp_penalties**: list[*glotaran.model.clp_penalties.ClpPenalty*]

clp_constraints

Model.**clp_constraints**: list[*glotaran.model.clp_constraint.ClpConstraint*]

clp_relations

Model.**clp_relations**: list[*glotaran.model.clp_relation.ClpRelation*]

dataset_groups

Model.**dataset_groups**: dict[str, *glotaran.model.dataset_group.DatasetGroupModel*]

dataset

Model.**dataset**: dict[str, *glotaran.model.dataset_model.DatasetModel*]

megacomplex

Model.**megacomplex**: dict[str, glotaran.model.megacomplex.Megacomplex]

weights

Model.**weights**: list[*glotaran.model.weight.Weight*]

Methods Summary

as_dict	Get the model as dictionary.
create_class	Create model class.
create_class_from_megacomplexes	Create model class for megacomplexes.
generate_parameters	Generate parameters for the model.
get_dataset_groups	Get the dataset groups.
get_issues	Get issues.
get_parameter_labels	Get all parameter labels.
iterate_all_items	Iterate the individual items.
iterate_items	Iterate items.
loader	Create a Model instance from the specs defined in a file.
markdown	Format the model as Markdown string.
valid	Check if the model is valid.
validate	Get a string listing all issues in the model and missing parameters if specified.

as_dict

Model.**as_dict**() → dict

Get the model as dictionary.

Return type
dict

create_class

classmethod Model.**create_class**(*attributes: dict[str, attr._make.Attribute]*) → type[*glotaran.model.model.Model*]

Create model class.

Parameters
attributes (*dict[str, Attribute]*) – The model attributes.

Return type
type[*Model*]

create_class_from_megacomplexes

classmethod Model.**create_class_from_megacomplexes**(*megacomplexes: Iterable[type[glotaran.model.megacomplex.Megacomplex]]*) → type[*glotaran.model.model.Model*]

Create model class for megacomplexes.

Parameters

megacomplexes (*list[type[Megacomplex]]*) – The megacomplexes.

Return type

type[*Model*]

generate_parameters

Model.**generate_parameters**() → *Parameters*

Generate parameters for the model.

Returns

- *Parameters* – The generated parameters.
- **.. # noqa** (*D414*)

get_dataset_groups

Model.**get_dataset_groups**() → dict[str, *glotaran.model.dataset_group.DatasetGroup*]

Get the dataset groups.

Return type

dict[str, *DatasetGroup*]

Raises

ModelError – Raised if a dataset group is unknown.

get_issues

Model.**get_issues**(*, *parameters:* Parameters | *None* = *None*) → list[glotaran.model.item.ItemIssue]

Get issues.

Parameters

parameters (Parameters | *None*) – The parameters.

Return type

list[ItemIssue]

get_parameter_labels

Model.**get_parameter_labels**() → set[str]

Get all parameter labels.

Return type

set[str]

iterate_all_items

Model.**iterate_all_items**() → Generator[Item, None, None]

Iterate the individual items.

Yields

Item – The individual item.

iterate_items

Model.**iterate_items**() → Generator[tuple[str, dict[str, glotaran.model.item.Item] | list[glotaran.model.item.Item]], None, None]

Iterate items.

Yields

tuple[str, dict[str, Item] | list[Item]] – The name of the item and the individual items of the type.

loader

Model.**loader**(*format_name: str | None = None*, ***kwargs: Any*) → *Model*

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns

Model instance created from the file.

Return type

Model

markdown

Model.**markdown**(*parameters: Parameters | None = None*, *initial_parameters: Parameters | None = None*, *base_heading_level: int = 1*) → *MarkdownStr*

Format the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameters** (*Parameters | None*) – Parameter to include.
- **initial_parameters** (*Parameters | None*) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:
  - If it is 1 the string will start with '# Model'.
  - If it is 3 the string will start with '### Model'.

Return type

MarkdownStr

valid

Model.**valid**(*parameters:* Parameters | *None* = *None*) → bool

Check if the model is valid.

Parameters
parameters (Parameters | *None*) – The parameters.

Return type
bool

validate

Model.**validate**(*parameters:* Parameters | *None* = *None*, *raise_exception: bool* = *False*) → *MarkdownStr*

Get a string listing all issues in the model and missing parameters if specified.

Parameters

- **parameters** (Parameters | *None*) – The parameters.
- **raise_exception** (*bool*) – Whether to raise an exception on failed validation.

Return type
MarkdownStr

Raises
ModelError – Raised if validation fails and raise_exception is true.

Methods Documentation

as_dict() → dict

Get the model as dictionary.

Return type
dict

clp_constraints: list[*glotaran.model.clp_constraint.ClpConstraint*]

clp_penalties: list[*glotaran.model.clp_penalties.ClpPenalty*]

clp_relations: list[*glotaran.model.clp_relation.ClpRelation*]

classmethod create_class(*attributes: dict[str, attr._make.Attribute]*) → type[*glotaran.model.model.Model*]

Create model class.

Parameters
attributes (*dict[str, Attribute]*) – The model attributes.

Return type
type[*Model*]

classmethod create_class_from_megacomplexes(*megacomplexes: Iterable[type[glotaran.model.megacomplex.Megacomplex]]*) → type[*glotaran.model.model.Model*]

Create model class for megacomplexes.

Parameters
megacomplexes (*list[type[Megacomplex]]*) – The megacomplexes.

Return type
type[*Model*]

dataset: dict[str, *glotaran.model.dataset_model.DatasetModel*]

dataset_groups: dict[str, *glotaran.model.dataset_group.DatasetGroupModel*]

generate_parameters() → *Parameters*

Generate parameters for the model.

Returns

- *Parameters* – The generated parameters.
- **.. # noqa** (*D414*)

get_dataset_groups() → dict[str, *glotaran.model.dataset_group.DatasetGroup*]

Get the dataset groups.

Return type
dict[str, *DatasetGroup*]

Raises
ModelError – Raised if a dataset group is unknown.

get_issues(*, *parameters:* Parameters | *None* = *None*) → list[glotaran.model.item.ItemIssue]

Get issues.

Parameters
parameters (Parameters | *None*) – The parameters.

Return type
list[ItemIssue]

get_parameter_labels() → set[str]

Get all parameter labels.

Return type
set[str]

iterate_all_items() → Generator[Item, None, None]

Iterate the individual items.

Yields
Item – The individual item.

iterate_items() → Generator[tuple[str, dict[str, glotaran.model.item.Item] | list[glotaran.model.item.Item]], None, None]

Iterate items.

Yields
tuple[str, dict[str, Item] | list[Item]] – The name of the item and the individual items of the type.

loader(*format_name: str | None = None*, ***kwargs: Any*) → *Model*

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns
Model instance created from the file.

Return type
Model

markdown(*parameters: Parameters | None = None*, *initial_parameters: Parameters | None = None*, *base_heading_level: int = 1*) → *MarkdownStr*

Format the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameters** (*Parameters | None*) – Parameter to include.
- **initial_parameters** (*Parameters | None*) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

Return type
MarkdownStr

megacomplex: dict[str, glotaran.model.megacomplex.Megacomplex]

source_path: str | None

valid(*parameters: Parameters | None = None*) → bool

Check if the model is valid.

Parameters
parameters (*Parameters | None*) – The parameters.

Return type
bool

validate(*parameters: Parameters | None = None*, *raise_exception: bool = False*) → *MarkdownStr*

Get a string listing all issues in the model and missing parameters if specified.

Parameters

- **parameters** (*Parameters | None*) – The parameters.

- **raise_exception** (*bool*) – Whether to raise an exception on failed validation.

Return type
MarkdownStr

Raises
ModelError – Raised if validation fails and raise_exception is true.

weights: list[*glotaran.model.weight.Weight*]

Exceptions

Exception Summary

ModelError	Raised when a model contains errors.

ModelError

exception glotaran.model.model.**ModelError**(*error: str*)

Raised when a model contains errors.

Create a model error.

Parameters
error (*str*) – The error string.

weight

This module contains weight item.

Classes

Summary

Weight	The *Weight* class describes a value by which a dataset will scaled.

Weight

class glotaran.model.weight.**Weight**(*, *datasets: list[str]*, *global_interval: tuple[float, float] | None = None*, *model_interval: tuple[float, float] | None = None*, *value: float*)

Bases: Item

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.

Method generated by attrs for class Weight.

Attributes Summary

datasets
global_interval
model_interval
value

datasets

Weight.**datasets**: list[str]

global_interval

Weight.**global_interval**: tuple[float, float] | None

model_interval

Weight.**model_interval**: tuple[float, float] | None

value

Weight.**value**: float

Methods Summary

Methods Documentation

datasets: list[str]

global_interval: tuple[float, float] | None

model_interval: tuple[float, float] | None

value: float

16.1.7 optimization

This package contains functions for optimization.

Modules

glotaran.optimization.data_provider	Module containing the data provider classes.
glotaran.optimization.estimation_provider	Module containing the estimation provider classes.
glotaran.optimization.matrix_provider	Module containing the matrix provider classes.
glotaran.optimization.nnls	Module for residual calculation with the non-negative least-squares method.
glotaran.optimization.optimization_group	Module containing the optimization group class.
glotaran.optimization.optimization_history	Module containing the `OptimizationHistory` class.
glotaran.optimization.optimize	Module containing the optimize function.
glotaran.optimization.optimizer	Module containing the optimizer class.
glotaran.optimization.variable_projection	Module for residual calculation with the variable projection method.

data_provider

Module containing the data provider classes.

Classes

Summary

DataProvider	A class to provide prepared data for optimization.
DataProviderLinked	A class to provide aligned data for optimization.

DataProvider

class glotaran.optimization.data_provider.**DataProvider**(*scheme:* Scheme, *dataset_group:* DatasetGroup)

Bases: `object`

A class to provide prepared data for optimization.

Initialize a data provider for a scheme and a dataset_group.

Parameters

- **scheme** (Scheme) – The optimization scheme.
- **dataset_group** (DatasetGroup) – The dataset group.

Methods Summary

add_model_weight	Add model weight to data.
get_axis_slice_from_interval	Get a slice of indices from a min max tuple and for an axis.
get_data	Get data for a dataset.
get_flattened_data	Get flattened data for a dataset.
get_flattened_weight	Get flattened weight for a dataset.
get_from_dataset	Get a copy of data from a dataset with dimensions (model, global).
get_global_axis	Get the global axis for a dataset.
get_global_dimension	Get the global dimension for a dataset.
get_model_axis	Get the model axis for a dataset.
get_model_dimension	Get the model dimension for a dataset.
get_weight	Get weight for a dataset.
infer_global_dimension	Infer the name of the global dimension from tuple of dimensions.

add_model_weight

DataProvider.**add_model_weight**(*model:* Model, *dataset_label:* *str*, *model_dimension:* *str*, *global_dimension:* *str*)

Add model weight to data.

Parameters

- **model** (Model) – The model.
- **dataset_label** (*str*) – The label of the data.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

get_axis_slice_from_interval

static DataProvider.**get_axis_slice_from_interval**(*interval:* *tuple[float, float]*, *axis:* *ArrayLike*) → slice

Get a slice of indices from a min max tuple and for an axis.

Parameters

- **interval** (*tuple[float, float]*) – The min max tuple.
- **axis** (*ArrayLike*) – The axis to slice.

Returns
The slice of indices.

Return type
slice

get_data

DataProvider.**get_data**(*dataset_label: str*) → ArrayLike

Get data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The data.

Return type
ArrayLike

get_flattened_data

DataProvider.**get_flattened_data**(*dataset_label: str*) → ArrayLike

Get flattened data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened data.

Return type
ArrayLike

get_flattened_weight

DataProvider.**get_flattened_weight**(*dataset_label: str*) → ArrayLike | None

Get flattened weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened weight.

Return type
ArrayLike | None

get_from_dataset

static DataProvider.**get_from_dataset**(*dataset: xr.Dataset*, *name: str*, *model_dimension: str*, *global_dimension: str*) → ArrayLike | None

Get a copy of data from a dataset with dimensions (model, global).

Parameters

- **dataset** (*xr.Dataset*) – The dataset to retrieve from.
- **name** (*str*) – The name of the data to retrieve.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

Returns
The copy of the data. None if name is not present in dataset.

Return type
ArrayLike | None

get_global_axis

DataProvider.**get_global_axis**(*dataset_label: str*) → ArrayLike

Get the global axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global axis.

Return type
ArrayLike

get_global_dimension

DataProvider.**get_global_dimension**(*dataset_label: str*) → str

Get the global dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global dimension.

Return type
str

get_model_axis

DataProvider.**get_model_axis**(*dataset_label: str*) → ArrayLike

Get the model axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model axis.

Return type
ArrayLike

get_model_dimension

DataProvider.**get_model_dimension**(*dataset_label: str*) → str

Get the model dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model dimension.

Return type
str

get_weight

DataProvider.**get_weight**(*dataset_label: str*) → ArrayLike | None

Get weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The weight.

Return type
ArrayLike | None

infer_global_dimension

static DataProvider.**infer_global_dimension**(*model_dimension: str*, *dimensions: tuple[str]*) → str

Infer the name of the global dimension from tuple of dimensions.

Parameters

- **model_dimension** (*str*) – The model dimension.
- **dimensions** (*tuple[str]*) – The dimensions tuple to infer from.

Returns
The inferred name of the global dimension.

Return type
str

Methods Documentation

add_model_weight(*model: Model*, *dataset_label: str*, *model_dimension: str*, *global_dimension: str*)

Add model weight to data.

Parameters

- **model** (*Model*) – The model.
- **dataset_label** (*str*) – The label of the data.

- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

static get_axis_slice_from_interval(*interval: tuple[float, float]*, *axis: ArrayLike*) → slice

Get a slice of indices from a min max tuple and for an axis.

Parameters

- **interval** (*tuple[float, float]*) – The min max tuple.
- **axis** (*ArrayLike*) – The axis to slice.

Returns
The slice of indices.

Return type
slice

get_data(*dataset_label: str*) → ArrayLike

Get data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The data.

Return type
ArrayLike

get_flattened_data(*dataset_label: str*) → ArrayLike

Get flattened data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened data.

Return type
ArrayLike

get_flattened_weight(*dataset_label: str*) → ArrayLike | None

Get flattened weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened weight.

Return type
ArrayLike | None

static get_from_dataset(*dataset: xr.Dataset*, *name: str*, *model_dimension: str*, *global_dimension: str*) → ArrayLike | None

Get a copy of data from a dataset with dimensions (model, global).

Parameters

- **dataset** (*xr.Dataset*) – The dataset to retrieve from.

- **name** (*str*) – The name of the data to retrieve.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

Returns
The copy of the data. None if name is not present in dataset.

Return type
ArrayLike | None

get_global_axis(*dataset_label: str*) → ArrayLike

Get the global axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global axis.

Return type
ArrayLike

get_global_dimension(*dataset_label: str*) → str

Get the global dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global dimension.

Return type
str

get_model_axis(*dataset_label: str*) → ArrayLike

Get the model axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model axis.

Return type
ArrayLike

get_model_dimension(*dataset_label: str*) → str

Get the model dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model dimension.

Return type
str

get_weight(*dataset_label: str*) → ArrayLike | None

Get weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The weight.

Return type
ArrayLike | None

static infer_global_dimension(*model_dimension: str*, *dimensions: tuple[str]*) → str

Infer the name of the global dimension from tuple of dimensions.

Parameters

- **model_dimension** (*str*) – The model dimension.
- **dimensions** (*tuple[str]*) – The dimensions tuple to infer from.

Returns
The inferred name of the global dimension.

Return type
str

DataProviderLinked

class glotaran.optimization.data_provider.**DataProviderLinked**(*scheme: Scheme*, *dataset_group: DatasetGroup*)

Bases: *DataProvider*

A class to provide aligned data for optimization.

Initialize a linked data provider for a scheme and a dataset_group.

Parameters

- **scheme** (Scheme) – The optimization scheme.
- **dataset_group** (DatasetGroup) – The dataset group.

Attributes Summary

aligned_global_axis	Get the aligned global axis for the dataset group.
group_definitions	Get the group definitions for the dataset group.

aligned_global_axis

DataProviderLinked.**aligned_global_axis**

Get the aligned global axis for the dataset group.

Returns
The aligned global axis.

Return type
ArrayLike

group_definitions

DataProviderLinked.**group_definitions**

Get the group definitions for the dataset group.

Returns
The group definitions.

Return type
dict[str, list[str]]

Methods Summary

add_model_weight	Add model weight to data.
align_data	Align the data in a dataset group.
align_dataset_indices	Align the global indices in a dataset group.
align_groups	Align the groups in a dataset group.
align_index	Align an index on a target axis.
align_weights	Align the weights in a dataset group.
create_aligned_global_axes	Create aligned global axes for the dataset group.
get_aligned_data	Get the aligned data for an index.
get_aligned_dataset_indices	Get the aligned dataset indices for an index.
get_aligned_group_label	Get the group label for an index.
get_aligned_weight	Get the aligned weight for an index.
get_axis_slice_from_interval	Get a slice of indices from a min max tuple and for an axis.
get_data	Get data for a dataset.
get_flattened_data	Get flattened data for a dataset.
get_flattened_weight	Get flattened weight for a dataset.
get_from_dataset	Get a copy of data from a dataset with dimensions (model, global).
get_global_axis	Get the global axis for a dataset.
get_global_dimension	Get the global dimension for a dataset.
get_model_axis	Get the model axis for a dataset.
get_model_dimension	Get the model dimension for a dataset.
get_weight	Get weight for a dataset.
infer_global_dimension	Infer the name of the global dimension from tuple of dimensions.

add_model_weight

DataProviderLinked.**add_model_weight**(*model:* Model, *dataset_label: str*, *model_dimension: str*, *global_dimension: str*)

Add model weight to data.

Parameters

- **model** (Model) – The model.
- **dataset_label** (*str*) – The label of the data.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

align_data

DataProviderLinked.**align_data**(*aligned_global_axes: dict[str, ArrayLike]*) → tuple[ArrayLike, list[ArrayLike]]

Align the data in a dataset group.

Parameters
aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns
The aligned global axis and data.

Return type
tuple[ArrayLike, list[ArrayLike]]

align_dataset_indices

DataProviderLinked.**align_dataset_indices**(*aligned_global_axes: dict[str, ArrayLike]*) → list[ArrayLike]

Align the global indices in a dataset group.

Parameters
aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns

- *list[ArrayLike]*
- *The aligned dataset indices.*

align_groups

static DataProviderLinked.**align_groups**(*aligned_global_axes: dict[str, ArrayLike]*) → tuple[ArrayLike, dict[str, list[str]]]

Align the groups in a dataset group.

Parameters
aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns
The aligned grouplabels and group definitions.

Return type
tuple[ArrayLike, dict[str, list[str]]]

align_index

static DataProviderLinked.**align_index**(*index: int*, *target_axis: ArrayLike*, *tolerance: float*, *method: Literal['nearest', 'backward', 'forward']*) → int

Align an index on a target axis.

Parameters

- **index** (*int*) – The index to align.
- **target_axis** (*ArrayLike*) – The axis to align the index on.
- **tolerance** (*float*) – The alignment tolerance.
- **method** (*Literal["nearest", "backward", "forward"]*) – The alignment method.

Returns
The aligned index.

Return type
int

align_weights

DataProviderLinked.**align_weights**(*aligned_global_axes: dict[str, ArrayLike]*) → list[ArrayLike | None]

Align the weights in a dataset group.

Parameters
aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns
The aligned weights.

Return type
list[ArrayLike | None]

create_aligned_global_axes

DataProviderLinked.**create_aligned_global_axes**(*scheme: Scheme*) → dict[str, ArrayLike]

Create aligned global axes for the dataset group.

Parameters
scheme (Scheme) – The optimization scheme.

Returns
The aligned global axes.

Return type
dict[str, ArrayLike]

Raises
AlignDatasetError – Raised when dataset alignment is ambiguous.

get_aligned_data

DataProviderLinked.**get_aligned_data**(*index: int*) → ArrayLike

Get the aligned data for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned data.

Return type
ArrayLike

get_aligned_dataset_indices

DataProviderLinked.**get_aligned_dataset_indices**(*index: int*) → ArrayLike

Get the aligned dataset indices for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned dataset indices.

Return type
ArrayLike

get_aligned_group_label

DataProviderLinked.**get_aligned_group_label**(*index: int*) → str

Get the group label for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned group label.

Return type
str

get_aligned_weight

DataProviderLinked.**get_aligned_weight**(*index: int*) → ArrayLike | None

Get the aligned weight for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned weight.

Return type
ArrayLike | None

get_axis_slice_from_interval

static DataProviderLinked.**get_axis_slice_from_interval**(*interval: tuple[float, float]*, *axis: ArrayLike*) → slice

Get a slice of indices from a min max tuple and for an axis.

Parameters

- **interval** (*tuple[float, float]*) – The min max tuple.
- **axis** (*ArrayLike*) – The axis to slice.

Returns
The slice of indices.

Return type
slice

get_data

DataProviderLinked.**get_data**(*dataset_label: str*) → ArrayLike

Get data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The data.

Return type
ArrayLike

get_flattened_data

DataProviderLinked.**get_flattened_data**(*dataset_label: str*) → ArrayLike

Get flattened data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened data.

Return type
ArrayLike

get_flattened_weight

DataProviderLinked.**get_flattened_weight**(*dataset_label: str*) → ArrayLike | None

Get flattened weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened weight.

Return type
ArrayLike | None

get_from_dataset

static DataProviderLinked.**get_from_dataset**(*dataset: xr.Dataset*, *name: str*, *model_dimension: str*, *global_dimension: str*) → ArrayLike | None

Get a copy of data from a dataset with dimensions (model, global).

Parameters

- **dataset** (*xr.Dataset*) – The dataset to retrieve from.
- **name** (*str*) – The name of the data to retrieve.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

Returns
The copy of the data. None if name is not present in dataset.

Return type
ArrayLike | None

get_global_axis

DataProviderLinked.**get_global_axis**(*dataset_label: str*) → ArrayLike

Get the global axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global axis.

Return type
ArrayLike

get_global_dimension

DataProviderLinked.**get_global_dimension**(*dataset_label: str*) → str

Get the global dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global dimension.

Return type
str

get_model_axis

DataProviderLinked.**get_model_axis**(*dataset_label: str*) → ArrayLike

Get the model axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model axis.

Return type
ArrayLike

get_model_dimension

DataProviderLinked.**get_model_dimension**(*dataset_label: str*) → str

Get the model dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model dimension.

Return type
str

get_weight

DataProviderLinked.**get_weight**(*dataset_label: str*) → ArrayLike | None

Get weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The weight.

Return type
ArrayLike | None

infer_global_dimension

static DataProviderLinked.**infer_global_dimension**(*model_dimension: str*, *dimensions: tuple[str]*) → str

Infer the name of the global dimension from tuple of dimensions.

Parameters

- **model_dimension** (*str*) – The model dimension.
- **dimensions** (*tuple[str]*) – The dimensions tuple to infer from.

Returns
The inferred name of the global dimension.

Return type
str

Methods Documentation

add_model_weight(*model: Model*, *dataset_label: str*, *model_dimension: str*, *global_dimension: str*)

Add model weight to data.

Parameters

- **model** (*Model*) – The model.
- **dataset_label** (*str*) – The label of the data.
- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

align_data(*aligned_global_axes: dict[str, ArrayLike]*) → tuple[ArrayLike, list[ArrayLike]]

Align the data in a dataset group.

Parameters
aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns
The aligned global axis and data.

Return type
tuple[ArrayLike, list[ArrayLike]]

align_dataset_indices(*aligned_global_axes: dict[str, ArrayLike]*) → list[ArrayLike]

Align the global indices in a dataset group.

Parameters

aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns

- *list[ArrayLike]*
- *The aligned dataset indices.*

static align_groups(*aligned_global_axes: dict[str, ArrayLike]*) → tuple[ArrayLike, dict[str, list[str]]]

Align the groups in a dataset group.

Parameters

aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns

The aligned grouplabels and group definitions.

Return type

tuple[ArrayLike, dict[str, list[str]]]

static align_index(*index: int*, *target_axis: ArrayLike*, *tolerance: float*, *method: Literal['nearest', 'backward', 'forward']*) → int

Align an index on a target axis.

Parameters

- **index** (*int*) – The index to align.
- **target_axis** (*ArrayLike*) – The axis to align the index on.
- **tolerance** (*float*) – The alignment tolerance.
- **method** (*Literal["nearest", "backward", "forward"]*) – The alignment method.

Returns

The aligned index.

Return type

int

align_weights(*aligned_global_axes: dict[str, ArrayLike]*) → list[ArrayLike | None]

Align the weights in a dataset group.

Parameters

aligned_global_axes (*dict[str, ArrayLike]*) – The aligned global axes.

Returns

The aligned weights.

Return type

list[ArrayLike | None]

property aligned_global_axis: ArrayLike

Get the aligned global axis for the dataset group.

Returns

The aligned global axis.

Return type
ArrayLike

`create_aligned_global_axes`(*scheme: Scheme*) → dict[str, ArrayLike]

Create aligned global axes for the dataset group.

Parameters
scheme (Scheme) – The optimization scheme.

Returns
The aligned global axes.

Return type
dict[str, ArrayLike]

Raises
AlignDatasetError – Raised when dataset alignment is ambiguous.

`get_aligned_data`(*index: int*) → ArrayLike

Get the aligned data for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned data.

Return type
ArrayLike

`get_aligned_dataset_indices`(*index: int*) → ArrayLike

Get the aligned dataset indices for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned dataset indices.

Return type
ArrayLike

`get_aligned_group_label`(*index: int*) → str

Get the group label for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned group label.

Return type
str

`get_aligned_weight`(*index: int*) → ArrayLike | None

Get the aligned weight for an index.

Parameters
index (*int*) – The index on the aligned global axis.

Returns
The aligned weight.

Return type
ArrayLike | None

static get_axis_slice_from_interval(*interval: tuple[float, float]*, *axis: ArrayLike*) → slice

Get a slice of indices from a min max tuple and for an axis.

Parameters

- **interval** (*tuple[float, float]*) – The min max tuple.
- **axis** (*ArrayLike*) – The axis to slice.

Returns
The slice of indices.

Return type
slice

get_data(*dataset_label: str*) → ArrayLike

Get data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The data.

Return type
ArrayLike

get_flattened_data(*dataset_label: str*) → ArrayLike

Get flattened data for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened data.

Return type
ArrayLike

get_flattened_weight(*dataset_label: str*) → ArrayLike | None

Get flattened weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The flattened weight.

Return type
ArrayLike | None

static get_from_dataset(*dataset: xr.Dataset*, *name: str*, *model_dimension: str*, *global_dimension: str*) → ArrayLike | None

Get a copy of data from a dataset with dimensions (model, global).

Parameters

- **dataset** (*xr.Dataset*) – The dataset to retrieve from.
- **name** (*str*) – The name of the data to retrieve.

- **model_dimension** (*str*) – The model dimension.
- **global_dimension** (*str*) – The global dimension.

Returns
The copy of the data. None if name is not present in dataset.

Return type
ArrayLike | None

get_global_axis(*dataset_label: str*) → ArrayLike

Get the global axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global axis.

Return type
ArrayLike

get_global_dimension(*dataset_label: str*) → str

Get the global dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The global dimension.

Return type
str

get_model_axis(*dataset_label: str*) → ArrayLike

Get the model axis for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model axis.

Return type
ArrayLike

get_model_dimension(*dataset_label: str*) → str

Get the model dimension for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The model dimension.

Return type
str

get_weight(*dataset_label: str*) → ArrayLike | None

Get weight for a dataset.

Parameters
dataset_label (*str*) – The label of the data.

Returns
The weight.

Return type
ArrayLike | None

property group_definitions: dict[str, list[str]]

Get the group definitions for the dataset group.

Returns
The group definitions.

Return type
dict[str, list[str]]

static infer_global_dimension(*model_dimension: str*, *dimensions: tuple[str]*) → str

Infer the name of the global dimension from tuple of dimensions.

Parameters

- **model_dimension** (*str*) – The model dimension.
- **dimensions** (*tuple[str]*) – The dimensions tuple to infer from.

Returns
The inferred name of the global dimension.

Return type
str

Exceptions

Exception Summary

AlignDatasetError	Indicates that datasets can not be aligned.

AlignDatasetError

exception glotaran.optimization.data_provider.**AlignDatasetError**

Indicates that datasets can not be aligned.

Initialize a AlignDatasetError.

estimation_provider

Module containing the estimation provider classes.

Classes

Summary

EstimationProvider	A class to provide estimation for optimization.
EstimationProviderLinked	A class to provide estimation for optimization of a linked dataset group.
EstimationProviderUnlinked	A class to provide estimation for optimization of an unlinked dataset group.

EstimationProvider

class glotaran.optimization.estimation_provider.**EstimationProvider**(*dataset_group:* DatasetGroup)

Bases: object

A class to provide estimation for optimization.

Initialize an estimation provider for a dataset group.

Parameters
dataset_group (DatasetGroup) – The dataset group.

Raises
UnsupportedResidualFunctionError – Raised when residual function of the group dataset group is unsupported.

Attributes Summary

group	Get the dataset group.

group

EstimationProvider.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

Methods Summary

calculate_clp_penalties	Calculate the clp penalty.
calculate_residual	Calculate the clps and the residual for a matrix and data.
estimate	Calculate the estimation.
get_additional_penalties	Get the additional penalty.
get_full_penalty	Get the full penalty.
get_result	Get the results of the estimation.
retrieve_clps	Retrieve clp from reduced clp.

calculate_clp_penalties

EstimationProvider.**calculate_clp_penalties**(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.
- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_residual

EstimationProvider.**calculate_residual**(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate

EstimationProvider.**estimate**()

Calculate the estimation.

get_additional_penalties

EstimationProvider.**get_additional_penalties**() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty

EstimationProvider.**get_full_penalty**() → ArrayLike

Get the full penalty.

Returns

- *ArrayLike* – The clp penalty.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

get_result

EstimationProvider.**get_result**() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the estimation.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the estimated clps and residuals.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

retrieve_clps

EstimationProvider.**retrieve_clps**(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.
- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.

- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

Methods Documentation

calculate_clp_penalties(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.
- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_residual(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate()

Calculate the estimation.

get_additional_penalties() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty() → ArrayLike

Get the full penalty.

Returns

- *ArrayLike* – The clp penalty.

- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

get_result() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the estimation.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the estimated clps and residuals.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

property group: *DatasetGroup*

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

retrieve_clps(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.
- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.
- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

EstimationProviderLinked

class glotaran.optimization.estimation_provider.**EstimationProviderLinked**(*dataset_group:* DatasetGroup, *data_provider:* DataProviderLinked, *matrix_provider:* MatrixProviderLinked)

Bases: *EstimationProvider*

A class to provide estimation for optimization of a linked dataset group.

Initialize an estimation provider for a linked dataset group.

Parameters

- **dataset_group** (DatasetGroup) – The dataset group.
- **data_provider** (DataProviderLinked) – The data provider.
- **matrix_provider** (MatrixProviderLinked) – The matrix provider.

Attributes Summary

group	Get the dataset group.

group

EstimationProviderLinked.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

Methods Summary

calculate_clp_penalties	Calculate the clp penalty.
calculate_residual	Calculate the clps and the residual for a matrix and data.
estimate	Calculate the estimation.
get_additional_penalties	Get the additional penalty.
get_full_penalty	Get the full penalty.
get_result	Get the results of the estimation.
retrieve_clps	Retrieve clp from reduced clp.

calculate_clp_penalties

EstimationProviderLinked.**calculate_clp_penalties**(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.

- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_residual

EstimationProviderLinked.**calculate_residual**(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate

EstimationProviderLinked.**estimate**()

Calculate the estimation.

get_additional_penalties

EstimationProviderLinked.**get_additional_penalties**() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty

EstimationProviderLinked.**get_full_penalty**() → ArrayLike

Get the full penalty.

Returns
The clp penalty.

Return type
ArrayLike

get_result

EstimationProviderLinked.**get_result**() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the estimation.

Returns
A tuple of the estimated clps and residuals.

Return type
tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]

retrieve_clps

EstimationProviderLinked.**retrieve_clps**(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.
- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.
- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

Methods Documentation

calculate_clp_penalties(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.
- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_residual(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate()

Calculate the estimation.

get_additional_penalties() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty() → ArrayLike

Get the full penalty.

Returns
The clp penalty.

Return type
ArrayLike

get_result() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the estimation.

Returns
A tuple of the estimated clps and residuals.

Return type
tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]

property group: *DatasetGroup*

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

retrieve_clps(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.

- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.
- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

EstimationProviderUnlinked

class glotaran.optimization.estimation_provider.**EstimationProviderUnlinked**(*dataset_group: DatasetGroup*, *data_provider: DataProvider*, *matrix_provider: MatrixProviderUnlinked*)

Bases: *EstimationProvider*

A class to provide estimation for optimization of an unlinked dataset group.

Initialize an estimation provider for an unlinked dataset group.

Parameters

- **dataset_group** (DatasetGroup) – The dataset group.
- **data_provider** (DataProvider) – The data provider.
- **matrix_provider** (MatrixProviderUnlinked) – The matrix provider.

Attributes Summary

group	Get the dataset group.

group

EstimationProviderUnlinked.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

Methods Summary

calculate_clp_penalties	Calculate the clp penalty.
calculate_estimation	Calculate the estimation for a dataset.
calculate_full_model_estimation	Calculate the estimation for a dataset with a full model.
calculate_residual	Calculate the clps and the residual for a matrix and data.
estimate	Calculate the estimation.
get_additional_penalties	Get the additional penalty.
get_full_penalty	Get the full penalty.
get_result	Get the results of the estimation.
retrieve_clps	Retrieve clp from reduced clp.

calculate_clp_penalties

EstimationProviderUnlinked.**calculate_clp_penalties**(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.
- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_estimation

EstimationProviderUnlinked.**calculate_estimation**(*dataset_model:* DatasetModel)

Calculate the estimation for a dataset.

Parameters
dataset_model (DatasetModel) – The dataset model.

calculate_full_model_estimation

EstimationProviderUnlinked.**calculate_full_model_estimation**(*dataset_model:* DatasetModel)

Calculate the estimation for a dataset with a full model.

Parameters
dataset_model (DatasetModel) – The dataset model.

calculate_residual

EstimationProviderUnlinked.**calculate_residual**(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate

EstimationProviderUnlinked.**estimate**()

Calculate the estimation.

get_additional_penalties

EstimationProviderUnlinked.**get_additional_penalties**() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty

EstimationProviderUnlinked.**get_full_penalty**() → ArrayLike

Get the full penalty.

Returns
The clp penalty.

Return type
ArrayLike

get_result

EstimationProviderUnlinked.**get_result**() → tuple[dict[str, list[xarray.core.dataarray.DataArray]], dict[str, list[xarray.core.dataarray.DataArray]]]

Get the results of the estimation.

Returns
A tuple of the estimated clps and residuals.

Return type
tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]

retrieve_clps

EstimationProviderUnlinked.**retrieve_clps**(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.
- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.
- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

Methods Documentation

calculate_clp_penalties(*clp_labels: list[list[str]]*, *clps: list[numpy.ndarray]*, *global_axis: ndarray*) → list[float]

Calculate the clp penalty.

Parameters

- **clp_labels** (*list[list[str]]*) – The clp labels.
- **clps** (*list[ArrayLike]*) – The clps.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The clp penalty.

Return type
list[float]

calculate_estimation(*dataset_model:* DatasetModel)

Calculate the estimation for a dataset.

Parameters
dataset_model (DatasetModel) – The dataset model.

calculate_full_model_estimation(*dataset_model:* DatasetModel)

Calculate the estimation for a dataset with a full model.

Parameters
dataset_model (DatasetModel) – The dataset model.

calculate_residual(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

Calculate the clps and the residual for a matrix and data.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **data** (*ArrayLike*) – The data.

Returns
The estimated clp and residual.

Return type
tuple[ArrayLike, ArrayLike]

estimate()

Calculate the estimation.

get_additional_penalties() → list[float]

Get the additional penalty.

Returns
The additional penalty.

Return type
list[float]

get_full_penalty() → ArrayLike

Get the full penalty.

Returns
The clp penalty.

Return type
ArrayLike

get_result() → tuple[dict[str, list[xarray.core.dataarray.DataArray]], dict[str, list[xarray.core.dataarray.DataArray]]]

Get the results of the estimation.

Returns
A tuple of the estimated clps and residuals.

Return type
tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]

property group: *DatasetGroup*

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

retrieve_clps(*clp_labels: list[str]*, *reduced_clp_labels: list[str]*, *reduced_clps: ArrayLike*, *index: int*) → ArrayLike

Retrieve clp from reduced clp.

Parameters

- **clp_labels** (*list[str]*) – The original clp labels.
- **reduced_clp_labels** (*list[str]*) – The reduced clp labels.
- **reduced_clps** (*ArrayLike*) – The reduced clps.
- **index** (*int*) – The index on the global axis.

Returns
The retrieved clps.

Return type
ArrayLike

Exceptions

Exception Summary

UnsupportedResidualFunctionError	Inidcates that the residual function is unsupported.

UnsupportedResidualFunctionError

exception glotaran.optimization.estimation_provider.**UnsupportedResidualFunctionError**(*residual_function: str*)

Inidcates that the residual function is unsupported.

Initialize an UnsupportedMethodError.

Parameters
residual_function (*str*) – The unsupported residual_function.

matrix_provider

Module containing the matrix provider classes.

Classes

Summary

MatrixContainer	A container of matrix and the corresponding clp labels.
MatrixProvider	A class to provide matrix calculations for optimization.
MatrixProviderLinked	A class to provide matrix calculations for optimization of linked dataset groups.
MatrixProviderUnlinked	A class to provide matrix calculations for optimization of unlinked dataset groups.

MatrixContainer

class glotaran.optimization.matrix_provider.**MatrixContainer**(*clp_labels: list[str]*, *matrix: ndarray*)

Bases: object

A container of matrix and the corresponding clp labels.

Attributes Summary

is_index_dependent	Check if the matrix is index dependent.
clp_labels	The clp labels.
matrix	The matrix.

is_index_dependent

MatrixContainer.**is_index_dependent**

Check if the matrix is index dependent.

Returns
Whether the matrix is index dependent.

Return type
bool

clp_labels

MatrixContainer.**clp_labels: list[str]**

The clp labels.

matrix

MatrixContainer.**matrix: ndarray**

The matrix.

Methods Summary

apply_weight	Apply weight on a matrix.
create_scaled_matrix	Create a matrix container with a scaled matrix.
create_weighted_matrix	Create a matrix container with a weighted matrix.

apply_weight

static MatrixContainer.**apply_weight**(*matrix: ArrayLike*, *weight: ArrayLike*) → ArrayLike

Apply weight on a matrix.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **weight** (*ArrayLike*) – The weight.

Returns
The weighted matrix.

Return type
ArrayLike

create_scaled_matrix

MatrixContainer.**create_scaled_matrix**(*scale: float*) → *MatrixContainer*

Create a matrix container with a scaled matrix.

Parameters
scale (*float*) – The scale.

Returns
The scaled matrix.

Return type
MatrixContainer

create_weighted_matrix

MatrixContainer.**create_weighted_matrix**(*weight: ArrayLike*) → *MatrixContainer*

Create a matrix container with a weighted matrix.

Parameters
weight (*ArrayLike*) – The weight.

Returns
The weighted matrix.

Return type
MatrixContainer

Methods Documentation

static apply_weight(*matrix: ArrayLike*, *weight: ArrayLike*) → ArrayLike

Apply weight on a matrix.

Parameters

- **matrix** (*ArrayLike*) – The matrix.
- **weight** (*ArrayLike*) – The weight.

Returns
The weighted matrix.

Return type
ArrayLike

clp_labels: list[str]

The clp labels.

create_scaled_matrix(*scale: float*) → *MatrixContainer*

Create a matrix container with a scaled matrix.

Parameters
scale (*float*) – The scale.

Returns
The scaled matrix.

Return type
MatrixContainer

create_weighted_matrix(*weight: ArrayLike*) → *MatrixContainer*

Create a matrix container with a weighted matrix.

Parameters
weight (*ArrayLike*) – The weight.

Returns
The weighted matrix.

Return type
MatrixContainer

property is_index_dependent: bool

Check if the matrix is index dependent.

Returns
Whether the matrix is index dependent.

Return type
bool

matrix: ndarray

The matrix.

MatrixProvider

class glotaran.optimization.matrix_provider.**MatrixProvider**(*dataset_group:* DatasetGroup)

Bases: object

A class to provide matrix calculations for optimization.

Initialize a matrix provider for a dataset group.

Parameters
dataset_group (DatasetGroup) – The dataset group.

Attributes Summary

group	Get the dataset group.
number_of_clps	Return number of conditionally linear parameters.

group

MatrixProvider.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

number_of_clps

MatrixProvider.**number_of_clps**

Return number of conditionally linear parameters.

Raises
NotImplementedError – This property needs to be implemented by subclasses.

See also:

MatrixProviderUnlinked, *MatrixProviderLinked*

Methods Summary

apply_constraints	Apply constraints on a matrix.
apply_relations	Apply relations on a matrix.
calculate	Calculate the matrices for optimization.
calculate_dataset_matrices	Calculate the matrices of the datasets in the dataset group.
calculate_dataset_matrix	Calculate the matrix for a dataset on an index on the global axis.
combine_megacomplex_matrices	Calculate the matrix for a dataset on an index on the global axis.
does_interval_item_apply	Check if an interval item applies on an index.
get_matrix_container	Get the matrix container for a dataset on an index on the global axis.
get_result	Get the results of the matrix calculations.
reduce_matrix	Reduce a matrix.

apply_constraints

MatrixProvider.**apply_constraints**(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*]*,) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations

MatrixProvider.**apply_relations**(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*]*,) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate

MatrixProvider.**calculate**()

Calculate the matrices for optimization.

calculate_dataset_matrices

MatrixProvider.**calculate_dataset_matrices**()

Calculate the matrices of the datasets in the dataset group.

calculate_dataset_matrix

static MatrixProvider.**calculate_dataset_matrix**(*dataset_model:* DatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, *global_matrix: bool = False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

combine_megacomplex_matrices

static MatrixProvider.**combine_megacomplex_matrices**(*matrix_left: ArrayLike*, *matrix_right: ArrayLike*, *clp_labels_left: list[str]*, *clp_labels_right: list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns
The combined clp labels and matrix.

Return type
tuple[list[str], ArrayLike]

does_interval_item_apply

static MatrixProvider.**does_interval_item_apply**(*prop:* IntervalItem, *index: int | None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int | None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_matrix_container

MatrixProvider.**get_matrix_container**(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_result

MatrixProvider.**get_result**() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

reduce_matrix

MatrixProvider.**reduce_matrix**(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

Methods Documentation

apply_constraints(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate()

Calculate the matrices for optimization.

calculate_dataset_matrices()

Calculate the matrices of the datasets in the dataset group.

static calculate_dataset_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, *global_matrix: bool = False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

static combine_megacomplex_matrices(*matrix_left: ArrayLike*, *matrix_right: ArrayLike*, *clp_labels_left: list[str]*, *clp_labels_right: list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns
The combined clp labels and matrix.

Return type
tuple[list[str], ArrayLike]

static does_interval_item_apply(*prop: IntervalItem*, *index: int | None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int | None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_matrix_container(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_result() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

property group: *DatasetGroup*

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

property number_of_clps: int

Return number of conditionally linear parameters.

Raises
NotImplementedError – This property needs to be implemented by subclasses.

See also:

MatrixProviderUnlinked, *MatrixProviderLinked*

reduce_matrix(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

MatrixProviderLinked

class glotaran.optimization.matrix_provider.**MatrixProviderLinked**(*group:* DatasetGroup, *data_provider:* DataProviderLinked)

Bases: *MatrixProvider*

A class to provide matrix calculations for optimization of linked dataset groups.

Initialize a matrix provider for a linked dataset group.

Parameters

- **dataset_group** (DatasetGroup) – The dataset group.
- **data_provider** (DataProviderLinked) – The data provider.

Attributes Summary

aligned_full_clp_labels	Get the aligned full clp labels.
group	Get the dataset group.
number_of_clps	Return number of conditionally linear parameters.

aligned_full_clp_labels

MatrixProviderLinked.**aligned_full_clp_labels**

Get the aligned full clp labels.

Returns
The full aligned clp labels.

Return type
list[list[str]]

group

MatrixProviderLinked.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

number_of_clps

MatrixProviderLinked.**number_of_clps**

Return number of conditionally linear parameters.

Return type
int

Methods Summary

align_full_clp_labels	Align the unreduced clp labels.
align_matrices	Align matrices.
apply_constraints	Apply constraints on a matrix.
apply_relations	Apply relations on a matrix.
calculate	Calculate the matrices for optimization.
calculate_aligned_matrices	Calculate the aligned matrices of the dataset group.
calculate_dataset_matrices	Calculate the matrices of the datasets in the dataset group.
calculate_dataset_matrix	Calculate the matrix for a dataset on an index on the global axis.
combine_megacomplex_matrices	Calculate the matrix for a dataset on an index on the global axis.
does_interval_item_apply	Check if an interval item applies on an index.
get_aligned_matrix_container	Get the aligned matrix container for an index on the aligned global axis.
get_matrix_container	Get the matrix container for a dataset on an index on the global axis.
get_result	Get the results of the matrix calculations.
reduce_matrix	Reduce a matrix.

align_full_clp_labels

MatrixProviderLinked.**align_full_clp_labels**() → dict[str, list[str]]

Align the unreduced clp labels.

Returns
The aligned clp for every group.

Return type
dict[str, list[str]]

align_matrices

static MatrixProviderLinked.**align_matrices**(*matrices: list[glotaran.optimization.matrix_provider.MatrixContainer], scales: list[float]*) → *MatrixContainer*

Align matrices.

Parameters

- **matrices** (*list[MatrixContainer]*) – The matrices to align.
- **scales** (*list[float]*) – The scales of the matrices.

Returns
The aligned matrix container.

Return type
MatrixContainer

apply_constraints

MatrixProviderLinked.**apply_constraints**(*matrices: list[MatrixContainer]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[MatrixContainer],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations

MatrixProviderLinked.**apply_relations**(*matrices: list[MatrixContainer]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[MatrixContainer],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate

MatrixProviderLinked.**calculate**()

Calculate the matrices for optimization.

calculate_aligned_matrices

MatrixProviderLinked.**calculate_aligned_matrices**()

Calculate the aligned matrices of the dataset group.

calculate_dataset_matrices

MatrixProviderLinked.**calculate_dataset_matrices**()

Calculate the matrices of the datasets in the dataset group.

calculate_dataset_matrix

static MatrixProviderLinked.**calculate_dataset_matrix**(*dataset_model:* DatasetModel, *global_axis:* *ArrayLike*, *model_axis:* *ArrayLike*, *global_matrix:* *bool* = *False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

combine_megacomplex_matrices

static MatrixProviderLinked.**combine_megacomplex_matrices**(*matrix_left: ArrayLike*, *matrix_right:* *ArrayLike*, *clp_labels_left:* *list[str]*, *clp_labels_right:* *list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns
The combined clp labels and matrix.

Return type
tuple[list[str], ArrayLike]

does_interval_item_apply

static MatrixProviderLinked.**does_interval_item_apply**(*prop:* IntervalItem, *index:* *int* | *None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int* | *None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_aligned_matrix_container

MatrixProviderLinked.**get_aligned_matrix_container**(*global_index: int*) → *MatrixContainer*

Get the aligned matrix container for an index on the aligned global axis.

Parameters
global_index (*int*) – The index on the global axis.

Returns
The matrix container.

Return type
MatrixContainer

get_matrix_container

MatrixProviderLinked.**get_matrix_container**(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_result

MatrixProviderLinked.**get_result**() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

reduce_matrix

MatrixProviderLinked.**reduce_matrix**(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

Methods Documentation

align_full_clp_labels() → dict[str, list[str]]

Align the unreduced clp labels.

Returns
The aligned clp for every group.

Return type
dict[str, list[str]]

static align_matrices(*matrices:* *list[*glotaran.optimization.matrix_provider.MatrixContainer*]*, *scales:* *list[float]*) → *MatrixContainer*

Align matrices.

Parameters

- **matrices** (*list[*MatrixContainer*]*) – The matrices to align.
- **scales** (*list[float]*) – The scales of the matrices.

Returns
The aligned matrix container.

Return type
MatrixContainer

property aligned_full_clp_labels: list[list[str]]

Get the aligned full clp labels.

Returns
The full aligned clp labels.

Return type
list[list[str]]

apply_constraints(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*]*,) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*]*,) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate()

Calculate the matrices for optimization.

calculate_aligned_matrices()

Calculate the aligned matrices of the dataset group.

calculate_dataset_matrices()

Calculate the matrices of the datasets in the dataset group.

static calculate_dataset_matrix(*dataset_model:* DatasetModel, *global_axis: ArrayLike*, *model_axis: ArrayLike*, *global_matrix: bool = False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.

- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

static combine_megacomplex_matrices(*matrix_left: ArrayLike*, *matrix_right: ArrayLike*, *clp_labels_left: list[str]*, *clp_labels_right: list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns
The combined clp labels and matrix.

Return type
tuple[list[str], ArrayLike]

static does_interval_item_apply(*prop:* IntervalItem, *index: int | None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int | None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_aligned_matrix_container(*global_index: int*) → *MatrixContainer*

Get the aligned matrix container for an index on the aligned global axis.

Parameters
global_index (*int*) – The index on the global axis.

Returns
The matrix container.

Return type
MatrixContainer

get_matrix_container(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

`get_result()` → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

`property group: DatasetGroup`

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

`property number_of_clps: int`

Return number of conditionally linear parameters.

Return type
int

`reduce_matrix`(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

MatrixProviderUnlinked

`class glotaran.optimization.matrix_provider.MatrixProviderUnlinked`(*group:* DatasetGroup, *data_provider:* DataProvider)

Bases: *MatrixProvider*

A class to provide matrix calculations for optimization of unlinked dataset groups.

Initialize a matrix provider for an unlinked dataset group.

Parameters

- **dataset_group** (DatasetGroup) – The dataset group.
- **data_provider** (DataProvider) – The data provider.

Attributes Summary

group	Get the dataset group.
number_of_clps	Return number of conditionally linear parameters.

group

MatrixProviderUnlinked.**group**

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

number_of_clps

MatrixProviderUnlinked.**number_of_clps**

Return number of conditionally linear parameters.

Return type
int

Methods Summary

apply_constraints	Apply constraints on a matrix.
apply_relations	Apply relations on a matrix.
calculate	Calculate the matrices for optimization.
calculate_dataset_matrices	Calculate the matrices of the datasets in the dataset group.
calculate_dataset_matrix	Calculate the matrix for a dataset on an index on the global axis.
calculate_full_matrices	Calculate the full matrices of the datasets in the dataset group.
calculate_global_matrices	Calculate the global matrices of the datasets in the dataset group.
calculate_prepared_matrices	Calculate the prepared matrices of the datasets in the dataset group.
combine_megacomplex_matrices	Calculate the matrix for a dataset on an index on the global axis.
does_interval_item_apply	Check if an interval item applies on an index.
get_full_matrix	Get the full matrix of a dataset.
get_global_matrix_container	Get the global matrix container for a dataset.
get_matrix_container	Get the matrix container for a dataset on an index on the global axis.
get_prepared_matrix_container	Get the prepared matrix container for a dataset on an index on the global axis.
get_result	Get the results of the matrix calculations.
reduce_matrix	Reduce a matrix.

apply_constraints

MatrixProviderUnlinked.**apply_constraints**(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*]*,) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations

MatrixProviderUnlinked.**apply_relations**(*matrices: list[MatrixContainer]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[MatrixContainer],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate

MatrixProviderUnlinked.**calculate**()

Calculate the matrices for optimization.

calculate_dataset_matrices

MatrixProviderUnlinked.**calculate_dataset_matrices**()

Calculate the matrices of the datasets in the dataset group.

calculate_dataset_matrix

static MatrixProviderUnlinked.**calculate_dataset_matrix**(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, *global_matrix: bool = False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate_full_matrices

MatrixProviderUnlinked.**calculate_full_matrices**()

Calculate the full matrices of the datasets in the dataset group.

calculate_global_matrices

MatrixProviderUnlinked.**calculate_global_matrices**()

Calculate the global matrices of the datasets in the dataset group.

calculate_prepared_matrices

MatrixProviderUnlinked.**calculate_prepared_matrices**()

Calculate the prepared matrices of the datasets in the dataset group.

combine_megacomplex_matrices

static MatrixProviderUnlinked.**combine_megacomplex_matrices**(*matrix_left: ArrayLike*, *matrix_right: ArrayLike*, *clp_labels_left: list[str]*, *clp_labels_right: list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns

The combined clp labels and matrix.

Return type

tuple[list[str], ArrayLike]

does_interval_item_apply

static MatrixProviderUnlinked.**does_interval_item_apply**(*prop:* IntervalItem, *index:* *int* | *None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int* | *None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_full_matrix

MatrixProviderUnlinked.**get_full_matrix**(*dataset_label: str*) → ArrayLike

Get the full matrix of a dataset.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix.

Return type
ArrayLike

get_global_matrix_container

MatrixProviderUnlinked.**get_global_matrix_container**(*dataset_label: str*) → *MatrixContainer*

Get the global matrix container for a dataset.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_matrix_container

MatrixProviderUnlinked.**get_matrix_container**(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_prepared_matrix_container

MatrixProviderUnlinked.**get_prepared_matrix_container**(*dataset_label: str*, *global_index: int*) → *MatrixContainer*

Get the prepared matrix container for a dataset on an index on the global axis.

Parameters

- **dataset_label** (*str*) – The label of the dataset.
- **global_index** (*int*) – The index on the global axis.

Returns
The matrix container.

Return type
MatrixContainer

get_result

MatrixProviderUnlinked.**get_result**() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

reduce_matrix

MatrixProviderUnlinked.**reduce_matrix**(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

Methods Documentation

apply_constraints(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply constraints on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

apply_relations(*matrices: list[*MatrixContainer*]*, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Apply relations on a matrix.

Parameters

- **matrices** (*list[*MatrixContainer*],*) – The matrices.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate()

Calculate the matrices for optimization.

calculate_dataset_matrices()

Calculate the matrices of the datasets in the dataset group.

static calculate_dataset_matrix(*dataset_model: DatasetModel*, *global_axis: ArrayLike*, *model_axis: ArrayLike*, *global_matrix: bool = False*) → *MatrixContainer*

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **dataset_model** (DatasetModel) – The dataset model.
- **global_axis** (*ArrayLike*) – The global axis.
- **model_axis** (*ArrayLike*) – The model axis.
- **global_matrix** (*bool*) – Calculate the global megacomplexes if *True*.

Returns
The resulting matrix container.

Return type
MatrixContainer

calculate_full_matrices()

Calculate the full matrices of the datasets in the dataset group.

calculate_global_matrices()

Calculate the global matrices of the datasets in the dataset group.

calculate_prepared_matrices()

Calculate the prepared matrices of the datasets in the dataset group.

static combine_megacomplex_matrices(*matrix_left: ArrayLike*, *matrix_right: ArrayLike*, *clp_labels_left: list[str]*, *clp_labels_right: list[str]*) → tuple[list[str], ArrayLike]

Calculate the matrix for a dataset on an index on the global axis.

Parameters

- **matrix_left** (*ArrayLike*) – The left matrix.
- **matrix_right** (*ArrayLike*) – The right matrix.
- **clp_labels_left** (*list[str]*) – The left clp labels.
- **clp_labels_right** (*list[str]*) – The right clp labels.

Returns
The combined clp labels and matrix.

Return type
tuple[list[str], ArrayLike]

static does_interval_item_apply(*prop: IntervalItem*, *index: int | None*) → bool

Check if an interval item applies on an index.

Parameters

- **prop** (IntervalItem) – The interval property.
- **index** (*int | None*) – The index to check.

Returns
Whether the property applies.

Return type
bool

get_full_matrix(*dataset_label: str*) → ArrayLike

Get the full matrix of a dataset.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix.

Return type
ArrayLike

get_global_matrix_container(*dataset_label: str*) → *MatrixContainer*

Get the global matrix container for a dataset.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_matrix_container(*dataset_label: str*) → *MatrixContainer*

Get the matrix container for a dataset on an index on the global axis.

Parameters
dataset_label (*str*) – The label of the dataset.

Returns
The matrix container.

Return type
MatrixContainer

get_prepared_matrix_container(*dataset_label: str*, *global_index: int*) → *MatrixContainer*

Get the prepared matrix container for a dataset on an index on the global axis.

Parameters

- **dataset_label** (*str*) – The label of the dataset.
- **global_index** (*int*) – The index on the global axis.

Returns
The matrix container.

Return type
MatrixContainer

get_result() → tuple[dict[str, xarray.core.dataarray.DataArray], dict[str, xarray.core.dataarray.DataArray]]

Get the results of the matrix calculations.

Returns

- *tuple[dict[str, xr.DataArray], dict[str, xr.DataArray]]* – A tuple of the matrices and global matrices.
- **.. # noqa** (*DAR202*)

- **.. # noqa** (*DAR401*)

property group: *DatasetGroup*

Get the dataset group.

Returns
The dataset group.

Return type
DatasetGroup

property number_of_clps: int

Return number of conditionally linear parameters.

Return type
int

reduce_matrix(*matrix:* MatrixContainer, *global_axis: ArrayLike*) → list[*MatrixContainer*]

Reduce a matrix.

Applies constraints and relations.

Parameters

- **matrix** (MatrixContainer) – The matrix.
- **global_axis** (*ArrayLike*) – The global axis.

Returns
The resulting matrix container.

Return type
MatrixContainer

nnls

Module for residual calculation with the non-negative least-squares method.

Functions

Summary

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

residual_nnls

glotaran.optimization.nnls.**residual_nnls**(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

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

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

Parameters

- **matrix** (*ArrayLike*) – The model matrix.

- **data** (*ArrayLike*) – The data to analyze.

Returns
The clps and the residual.

Return type
tuple[ArrayLike, ArrayLike]

optimization_group

Module containing the optimization group class.

Classes

Summary

OptimizationGroup	A class to optimize a dataset group.

OptimizationGroup

class glotaran.optimization.optimization_group.**OptimizationGroup**(*scheme: Scheme*, *dataset_group: DatasetGroup*)

Bases: object

A class to optimize a dataset group.

Initialize an optimization group for a dataset group.

Parameters

- **scheme** (Scheme) – The optimization scheme.
- **dataset_group** (DatasetGroup) – The dataset group.

Attributes Summary

number_of_clps	Return number of conditionally linear parameters.

number_of_clps

OptimizationGroup.**number_of_clps**

Return number of conditionally linear parameters.

Return type
int

Methods Summary

add_svd_data	Add the SVD of a data matrix to a dataset.
add_weight_to_result_data	Add weight to result dataset if dataset is weighted.
calculate	Calculate the optimization group data.
create_result_data	Create resulting datasets.
get_additional_penalties	Get additional penalties.
get_full_penalty	Get the full penalty.

add_svd_data

static OptimizationGroup.**add_svd_data**(*name: str*, *dataset: Dataset*, *lsv_dim: str*, *rsv_dim: str*)

Add the SVD of a data matrix to a dataset.

Parameters

- **name** (*str*) – Name of the data matrix.
- **dataset** (*xr.Dataset*) – Dataset containing the data, which will be updated with the SVD values.
- **lsv_dim** (*str*) – The dimension name of the left singular vectors.
- **rsv_dim** (*str*) – The dimension name of the right singular vectors.

add_weight_to_result_data

OptimizationGroup.**add_weight_to_result_data**(*dataset_label: str*, *result_dataset: Dataset*)

Add weight to result dataset if dataset is weighted.

Parameters

- **dataset_label** (*str*) – The label of the data.
- **result_dataset** (*xr.Dataset*) – The label of the data.

calculate

OptimizationGroup.**calculate**(*parameters:* Parameters)

Calculate the optimization group data.

Parameters

parameters (Parameters) – The parameters.

create_result_data

OptimizationGroup.**create_result_data**() → dict[str, xarray.core.dataset.Dataset]

Create resulting datasets.

Returns
The datasets with the results.

Return type
dict[str, xr.Dataset]

get_additional_penalties

OptimizationGroup.**get_additional_penalties**() → list[float]

Get additional penalties.

Returns
The additional penalties.

Return type
list[float]

get_full_penalty

OptimizationGroup.**get_full_penalty**() → ArrayLike

Get the full penalty.

Returns
The full penalty.

Return type
ArrayLike

Methods Documentation

static add_svd_data(*name: str*, *dataset: Dataset*, *lsv_dim: str*, *rsv_dim: str*)

Add the SVD of a data matrix to a dataset.

Parameters

- **name** (*str*) – Name of the data matrix.
- **dataset** (*xr.Dataset*) – Dataset containing the data, which will be updated with the SVD values.
- **lsv_dim** (*str*) – The dimension name of the left singular vectors.
- **rsv_dim** (*str*) – The dimension name of the right singular vectors.

add_weight_to_result_data(*dataset_label: str*, *result_dataset: Dataset*)

Add weight to result dataset if dataset is weighted.

Parameters

- **dataset_label** (*str*) – The label of the data.
- **result_dataset** (*xr.Dataset*) – The label of the data.

calculate(*parameters:* Parameters)

Calculate the optimization group data.

Parameters
parameters (Parameters) – The parameters.

create_result_data() → dict[str, xarray.core.dataset.Dataset]

Create resulting datasets.

Returns
The datasets with the results.

Return type
dict[str, xr.Dataset]

get_additional_penalties() → list[float]

Get additional penalties.

Returns
The additional penalties.

Return type
list[float]

get_full_penalty() → ArrayLike

Get the full penalty.

Returns
The full penalty.

Return type
ArrayLike

property number_of_clps: int

Return number of conditionally linear parameters.

Return type
int

optimization_history

Module containing the `OptimizationHistory` class.

Classes

Summary

OptimizationHistory	Wrapped DataFrame to hold information of the optimization and behaves like a `DataFrame`.

OptimizationHistory

class glotaran.optimization.optimization_history.**OptimizationHistory**(*data=None*, *source_path: StrOrPath | None = None*)

Bases: object

Wrapped DataFrame to hold information of the optimization and behaves like a DataFrame.

Ref.: https://stackoverflow.com/a/65375904/3990615

Ensure DataFrame has the correct columns, is numeric and has iteration as index.

Attributes Summary

data	Underlying DataFrame which allows for autocomplete with static analyzers.

data

OptimizationHistory.**data**

Underlying DataFrame which allows for autocomplete with static analyzers.

Returns
DataFrame containing OptimizationHistory data.

Return type
pd.DataFrame

Methods Summary

from_csv	Read OptimizationHistory from file.
from_stdout_str	Create OptimizationHistory instance from optimize_stdout.
loader	Read OptimizationHistory from file.
to_csv	Write a OptimizationHistory to a CSV file and set source_path.

from_csv

classmethod OptimizationHistory.**from_csv**(*path: StrOrPath*) → *OptimizationHistory*

Read `OptimizationHistory` from file.

Parameters
path (*StrOrPath*) – The path to the csv file.

Returns
`OptimizationHistory` read from file.

Return type
OptimizationHistory

from_stdout_str

classmethod OptimizationHistory.**from_stdout_str**(*optimize_stdout: str*) → *OptimizationHistory*

Create `OptimizationHistory` instance from `optimize_stdout`.

Parameters
optimize_stdout (*str*) – SciPy optimization stdout string, read out via `TeeContext.read()`.

Returns
`OptimizationHistory` instance created by parsing `optimize_stdout`.

Return type
OptimizationHistory

loader

classmethod OptimizationHistory.**loader**(*path: StrOrPath*) → *OptimizationHistory*

Read `OptimizationHistory` from file.

Parameters
path (*StrOrPath*) – The path to the csv file.

Returns
`OptimizationHistory` read from file.

Return type
OptimizationHistory

to_csv

OptimizationHistory.**to_csv**(*path: StrOrPath*, *delimiter: str = ','*)

Write a `OptimizationHistory` to a CSV file and set `source_path`.

Parameters

- **path** (*StrOrPath*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

Methods Documentation

property data: DataFrame

Underlying `DataFrame` which allows for autocomplete with static analyzers.

Returns
`DataFrame` containing `OptimizationHistory` data.

Return type
pd.DataFrame

classmethod from_csv(*path: StrOrPath*) → *OptimizationHistory*

Read `OptimizationHistory` from file.

Parameters
path (*StrOrPath*) – The path to the csv file.

Returns
`OptimizationHistory` read from file.

Return type
OptimizationHistory

classmethod from_stdout_str(*optimize_stdout: str*) → *OptimizationHistory*

Create `OptimizationHistory` instance from `optimize_stdout`.

Parameters
optimize_stdout (*str*) – SciPy optimization stdout string, read out via `TeeContext.read()`.

Returns
`OptimizationHistory` instance created by parsing `optimize_stdout`.

Return type
OptimizationHistory

classmethod loader(*path: StrOrPath*) → *OptimizationHistory*

Read `OptimizationHistory` from file.

Parameters
path (*StrOrPath*) – The path to the csv file.

Returns
`OptimizationHistory` read from file.

Return type
OptimizationHistory

to_csv(*path: StrOrPath*, *delimiter: str = ','*)

Write a `OptimizationHistory` to a CSV file and set `source_path`.

Parameters

- **path** (*StrOrPath*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

optimize

Module containing the optimize function.

Functions

Summary

optimize	Optimize a scheme.

optimize

glotaran.optimization.optimize.**optimize**(*scheme:* Scheme, *verbose:* *bool* = *True*, *raise_exception:* *bool* = *False*) → *Result*

Optimize a scheme.

Parameters

- **scheme** (Scheme) – The optimization scheme.
- **verbose** (*bool*) – Deactivate printing of logs if *False*.
- **raise_exception** (*bool*) – Raise exceptions during optimizations instead of gracefully exiting if *True*.

Returns
The result of the optimization.

Return type
Result

optimizer

Module containing the optimizer class.

Classes

Summary

Optimizer	A class to optimize a scheme.

Optimizer

class glotaran.optimization.optimizer.**Optimizer**(*scheme:* Scheme, *verbose:* *bool* = *True*, *raise_exception:* *bool* = *False*)

Bases: `object`

A class to optimize a scheme.

Initialize an optimization group for a dataset group.

Parameters

- **scheme** (Scheme) – The optimization scheme.
- **verbose** (*bool*) – Deactivate printing of logs if *False*.
- **raise_exception** (*bool*) – Raise exceptions during optimizations instead of gracefully exiting if *True*.

Raises

- **MissingDatasetsError** – Raised if datasets are missing.
- **ParameterNotInitializedError** – Raised if the scheme parameters are *None*.
- **UnsupportedMethodError** – Raised if the optimization method is unsupported.

Methods Summary

calculate_covariance_matrix_and_stan	Calculate the covariance matrix and standard errors of the optimization.
calculate_penalty	Calculate the penalty of the scheme.
create_result	Create the result of the optimization.
get_current_optimization_iteration	Extract current iteration from `optimize_stdout`.
objective_function	Calculate the objective for the optimization.
optimize	Perform the optimization.

calculate_covariance_matrix_and_standard_errors

`Optimizer`.**calculate_covariance_matrix_and_standard_errors**(*jacobian: ArrayLike*, *root_mean_square_error:* *float*) → ArrayLike

Calculate the covariance matrix and standard errors of the optimization.

Parameters

- **jacobian** (*ArrayLike*) – The jacobian matrix.
- **root_mean_square_error** (*float*) – The root mean square error.

Returns

The covariance matrix.

Return type
ArrayLike

calculate_penalty

Optimizer.**calculate_penalty**() → ArrayLike

Calculate the penalty of the scheme.

Returns
The penalty.

Return type
ArrayLike

create_result

Optimizer.**create_result**() → *Result*

Create the result of the optimization.

Returns
The result of the optimization.

Return type
Result

Raises
InitialParameterError – Raised if the initial parameters could not be evaluated.

get_current_optimization_iteration

static Optimizer.**get_current_optimization_iteration**(*optimize_stdout: str*) → int

Extract current iteration from `optimize_stdout`.

Parameters
optimize_stdout (*str*) – SciPy optimization stdout string, read out via `TeeContext.read()`.

Returns
Current iteration (**0** if pattern did not match).

Return type
int

objective_function

Optimizer.**objective_function**(*parameters: ArrayLike*) → ArrayLike

Calculate the objective for the optimization.

Parameters
parameters (*ArrayLike*) – the parameters provided by the optimizer.

Returns
The objective for the optimizer.

Return type
ArrayLike

optimize

`Optimizer.optimize()`

Perform the optimization.

Raises
`Exception` – Raised if an exception occurs during optimization and raise_exception is *True*.

Methods Documentation

calculate_covariance_matrix_and_standard_errors(*jacobian: ArrayLike*, *root_mean_square_error: float*) → ArrayLike

Calculate the covariance matrix and standard errors of the optimization.

Parameters

- **jacobian** (*ArrayLike*) – The jacobian matrix.
- **root_mean_square_error** (*float*) – The root mean square error.

Returns
The covariance matrix.

Return type
ArrayLike

calculate_penalty() → ArrayLike

Calculate the penalty of the scheme.

Returns
The penalty.

Return type
ArrayLike

create_result() → *Result*

Create the result of the optimization.

Returns
The result of the optimization.

Return type
Result

Raises
InitialParameterError – Raised if the initial parameters could not be evaluated.

static get_current_optimization_iteration(*optimize_stdout: str*) → int

Extract current iteration from `optimize_stdout`.

Parameters
optimize_stdout (*str*) – SciPy optimization stdout string, read out via `TeeContext.read()`.

Returns
Current iteration (`0` if pattern did not match).

Return type
int

`objective_function`(*parameters: ArrayLike*) → ArrayLike

Calculate the objective for the optimization.

Parameters
parameters (*ArrayLike*) – the parameters provided by the optimizer.

Returns
The objective for the optimizer.

Return type
ArrayLike

`optimize`()

Perform the optimization.

Raises
Exception – Raised if an exception occurs during optimization and raise_exception is *True*.

Exceptions

Exception Summary

`InitialParameterError`	Indicates that initial parameters can not be evaluated.
`MissingDatasetsError`	Indicates that datasets are missing in the scheme.
`ParameterNotInitializedError`	Indicates that scheme parameters are not initialized.
`UnsupportedMethodError`	Indicates that the optimization method is unsupported.

InitialParameterError

exception glotaran.optimization.optimizer.**InitialParameterError**

Indicates that initial parameters can not be evaluated.

Initialize a InitialParameterError.

MissingDatasetsError

exception glotaran.optimization.optimizer.**MissingDatasetsError**(*missing_datasets: list[str]*)

Indicates that datasets are missing in the scheme.

Initialize a MissingDatasetsError.

Parameters
missing_datasets (*list[str]*) – The missing datasets.

ParameterNotInitializedError

exception glotaran.optimization.optimizer.**ParameterNotInitializedError**

Indicates that scheme parameters are not initialized.

Initialize a ParameterNotInitializedError.

UnsupportedMethodError

exception glotaran.optimization.optimizer.**UnsupportedMethodError**(*method: str*)

Indicates that the optimization method is unsupported.

Initialize an UnsupportedMethodError.

Parameters
method (*str*) – The unsupported method.

variable_projection

Module for residual calculation with the variable projection method.

Functions

Summary

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

residual_variable_projection

glotaran.optimization.variable_projection.**residual_variable_projection**(*matrix: ArrayLike*, *data: ArrayLike*) → tuple[ArrayLike, ArrayLike]

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

Parameters

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

Returns
The clps and the residual.

Return type
tuple[ArrayLike, ArrayLike]

16.1.8 parameter

The glotaran parameter package.

Modules

glotaran.parameter.parameter	The parameter class.
glotaran.parameter.parameter_history	The glotaran parameter history package.
glotaran.parameter.parameters	The parameters class.

parameter

The parameter class.

Functions

Summary

deserialize_options	Replace options keys in serialized format by attribute names.
serialize_options	Replace options keys with serialized format by attribute names.
set_transformed_expression	Set the transformed expression from an expression.
valid_label	Check if a label is a valid label for *Parameter*.

deserialize_options

glotaran.parameter.parameter.**deserialize_options**(*options: dict[str, Any]*) → dict[str, Any]

Replace options keys in serialized format by attribute names.

Parameters
options (*dict[str, Any]*) – The serialized options.

Returns
The deserialized options.

Return type
dict[str, Any]

serialize_options

glotaran.parameter.parameter.**serialize_options**(*options: dict[str, Any]*) → dict[str, Any]

Replace options keys with serialized format by attribute names.

Parameters
options (*dict[str, Any]*) – The options.

Returns
The serialized options.

Return type
dict[str, Any]

set_transformed_expression

glotaran.parameter.parameter.**set_transformed_expression**(*parameter: Parameter*, *attribute: Attribute*, *expression: str | None*)

Set the transformed expression from an expression.

Parameters

- **parameter** (Parameter) – The *Parameter* instance
- **attribute** (*Attribute*) – The label field.
- **expression** (*str | None*) – The expression value.

valid_label

glotaran.parameter.parameter.**valid_label**(*parameter: Parameter*, *attribute: Attribute*, *label: str*)

Check if a label is a valid label for *Parameter*.

Parameters

- **parameter** (Parameter) – The *Parameter* instance
- **attribute** (*Attribute*) – The label field.
- **label** (*str*) – The label value.

Raises
ValueError – Raise when the label is not valid.

Classes

Summary

Parameter	A parameter for optimization.

Parameter

class glotaran.parameter.parameter.**Parameter**(*label*, *value=nan*, *standard_error: float = nan*, *expression: str | None = None*, *maximum: float = inf*, *minimum: float = -inf*, *non_negative: bool = False*, *vary: bool = True*)

Bases: `_SupportsArray`

A parameter for optimization.

Method generated by attrs for class Parameter.

Attributes Summary

label	
value	
standard_error	
expression	
maximum	
minimum	
non_negative	
vary	
transformed_expression	
label_short	Get short label.

label

Parameter.**label**: str

value

Parameter.**value**: float

standard_error

Parameter.**standard_error**: float

expression

Parameter.**expression**: str | None

maximum

Parameter.**maximum**: float

minimum

Parameter.**minimum**: float

non_negative

Parameter.**non_negative**: bool

vary

Parameter.**vary**: bool

transformed_expression

Parameter.**transformed_expression**: str | None

label_short

Parameter.**label_short**

Get short label.

Returns
The short label.

Return type
str

Methods Summary

as_dict	Get the parameter as a dictionary.
as_list	Get the parameter as a dictionary.
copy	Create a copy of the *Parameter*.
from_list	Create a parameter from a list.
get_value_and_bounds_for_optimizatio	Get the parameter value and bounds with expression and non-negative constraints applied.
markdown	Get a markdown representation of the parameter.
set_value_from_optimization	Set the value from an optimization result and reverses non-negative transformation.

as_dict

Parameter.**as_dict**() → dict[str, Any]

Get the parameter as a dictionary.

Returns
The parameter as dictionary.

Return type
dict[str, Any]

as_list

Parameter.**as_list**(*label_short: bool = False*) → list[str | float | dict[str, Any]]

Get the parameter as a dictionary.

Parameters
label_short (*bool*) – If true, the label will be replaced by the shortened label.

Returns
The parameter as dictionary.

Return type
dict[str, Any]

copy

Parameter.**copy**() → *Parameter*

Create a copy of the *Parameter*.

Returns
A copy of the *Parameter*.

Return type
Parameter

from_list

classmethod Parameter.**from_list**(*values: list[Any]*, *, *default_options: dict[str, Any] | None = None*) → *Parameter*

Create a parameter from a list.

Parameters

- **values** (*list[Any]*) – The list of parameter definitions.
- **default_options** (*dict[str, Any] | None*) – A dictionary of default options.

Returns
The created *Parameter*.

Return type
Parameter

get_value_and_bounds_for_optimization

Parameter.**get_value_and_bounds_for_optimization**() → tuple[float, float, float]

Get the parameter value and bounds with expression and non-negative constraints applied.

Returns
A tuple containing the value, the lower and the upper bound.

Return type
tuple[float, float, float]

markdown

Parameter.**markdown**(*all_parameters:* Parameters | *None = None*, *initial_parameters:* Parameters | *None = None*) → *MarkdownStr*

Get a markdown representation of the parameter.

Parameters

- **all_parameters** (Parameters | *None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (Parameters | *None*) – The initial parameter.

Returns
The parameter as markdown string.

Return type
MarkdownStr

set_value_from_optimization

Parameter.**set_value_from_optimization**(*value: float*)

Set the value from an optimization result and reverses non-negative transformation.

Parameters
value (*float*) – Value from optimization.

Methods Documentation

as_dict() → dict[str, Any]

Get the parameter as a dictionary.

Returns
The parameter as dictionary.

Return type
dict[str, Any]

as_list(*label_short: bool = False*) → list[str | float | dict[str, Any]]

Get the parameter as a dictionary.

Parameters
label_short (*bool*) – If true, the label will be replaced by the shortened label.

Returns
The parameter as dictionary.

Return type
dict[str, Any]

copy() → *Parameter*

Create a copy of the *Parameter*.

Returns
A copy of the *Parameter*.

Return type
Parameter

expression: str | None

classmethod from_list(*values: list[Any]*, *, *default_options: dict[str, Any] | None = None*) → *Parameter*

Create a parameter from a list.

Parameters

- **values** (*list[Any]*) – The list of parameter definitions.
- **default_options** (*dict[str, Any] | None*) – A dictionary of default options.

Returns
The created *Parameter*.

Return type
Parameter

get_value_and_bounds_for_optimization() → tuple[float, float, float]

Get the parameter value and bounds with expression and non-negative constraints applied.

Returns
A tuple containing the value, the lower and the upper bound.

Return type
tuple[float, float, float]

label: str

property label_short: str

Get short label.

Returns
The short label.

Return type
str

markdown(*all_parameters:* Parameters | *None* = *None*, *initial_parameters:* Parameters | *None* = *None*) → *MarkdownStr*

Get a markdown representation of the parameter.

Parameters

- **all_parameters** (Parameters | *None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (Parameters | *None*) – The initial parameter.

Returns
The parameter as markdown string.

Return type
MarkdownStr

maximum: float

minimum: float

non_negative: bool

set_value_from_optimization(*value: float*)

Set the value from an optimization result and reverses non-negative transformation.

Parameters
value (*float*) – Value from optimization.

standard_error: float

transformed_expression: str | None

value: float

vary: bool

parameter_history

The glotaran parameter history package.

Classes

Summary

ParameterHistory	A class representing a history of parameters.

ParameterHistory

class glotaran.parameter.parameter_history.**ParameterHistory**

Bases: object

A class representing a history of parameters.

Attributes Summary

number_of_records	Return the number of records in the history.
parameter_labels	Return the labels of the parameters in the history.
parameters	Return the parameters in the history.

number_of_records

ParameterHistory.**number_of_records**

Return the number of records in the history.

Returns
The number of records.

Return type
int

parameter_labels

ParameterHistory.**parameter_labels**

Return the labels of the parameters in the history.

Returns
A list of parameter labels.

Return type
list[str]

parameters

ParameterHistory.**parameters**

Return the parameters in the history.

Returns
A list of parameters in the history.

Return type
list[np.ndarray]

Methods Summary

append	Append Parameters to the history.
from_csv	Create a history from a csv file.
from_dataframe	Create a history from a pandas data frame.
get_parameters	Get parameters for a history index.
loader	Create a history from a csv file.
to_csv	Write a *ParameterHistory* to a CSV file.
to_dataframe	Create a data frame from the history.

append

ParameterHistory.**append**(*parameters:* Parameters, *current_iteration:* *int* = *0*)

Append Parameters to the history.

Parameters

- **parameters** (Parameters) – The group to append.
- **current_iteration** (*int*) – Current iteration of the optimizer.

Raises
ValueError – Raised if the parameter labels differs from previous.

from_csv

classmethod ParameterHistory.**from_csv**(*path:* *str*) → *ParameterHistory*

Create a history from a csv file.

Parameters
path (*str*) – The path to the csv file.

Returns
The created history.

Return type
ParameterHistory

from_dataframe

classmethod ParameterHistory.**from_dataframe**(*history_df: DataFrame*) → *ParameterHistory*

Create a history from a pandas data frame.

Parameters
history_df (*pd.DataFrame*) – The source data frame.

Returns
The created history.

Return type
ParameterHistory

get_parameters

ParameterHistory.**get_parameters**(*index: int*) → ndarray

Get parameters for a history index.

Parameters
index (*int*) – The history index.

Returns
The parameter values at the history index as array.

Return type
np.ndarray

loader

classmethod ParameterHistory.**loader**(*path: str*) → *ParameterHistory*

Create a history from a csv file.

Parameters
path (*str*) – The path to the csv file.

Returns
The created history.

Return type
ParameterHistory

to_csv

ParameterHistory.**to_csv**(*file_name: str | PathLike[str]*, *delimiter: str = ','*)

Write a *ParameterHistory* to a CSV file.

Parameters

- **file_name** (*str*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe

ParameterHistory.**to_dataframe**() → DataFrame

Create a data frame from the history.

Returns
The created data frame.

Return type
pd.DataFrame

Methods Documentation

append(*parameters:* Parameters, *current_iteration: int = 0*)

Append `Parameters` to the history.

Parameters

- **parameters** (`Parameters`) – The group to append.
- **current_iteration** (*int*) – Current iteration of the optimizer.

Raises
ValueError – Raised if the parameter labels differs from previous.

classmethod from_csv(*path: str*) → *ParameterHistory*

Create a history from a csv file.

Parameters
path (*str*) – The path to the csv file.

Returns
The created history.

Return type
ParameterHistory

classmethod from_dataframe(*history_df: DataFrame*) → *ParameterHistory*

Create a history from a pandas data frame.

Parameters
history_df (*pd.DataFrame*) – The source data frame.

Returns
The created history.

Return type
ParameterHistory

get_parameters(*index: int*) → ndarray

Get parameters for a history index.

Parameters
index (*int*) – The history index.

Returns
The parameter values at the history index as array.

Return type
np.ndarray

classmethod loader(*path: str*) → *ParameterHistory*

Create a history from a csv file.

Parameters
path (*str*) – The path to the csv file.

Returns
The created history.

Return type
ParameterHistory

property number_of_records: int

Return the number of records in the history.

Returns
The number of records.

Return type
int

property parameter_labels: list[str]

Return the labels of the parameters in the history.

Returns
A list of parameter labels.

Return type
list[str]

property parameters: list[numpy.ndarray]

Return the parameters in the history.

Returns
A list of parameters in the history.

Return type
list[np.ndarray]

to_csv(*file_name: str | PathLike[str]*, *delimiter: str = ','*)

Write a *ParameterHistory* to a CSV file.

Parameters

- **file_name** (*str*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe() → DataFrame

Create a data frame from the history.

Returns
The created data frame.

Return type
pd.DataFrame

parameters

The parameters class.

Functions

Summary

flatten_parameter_dict	Flatten a parameter dictionary.
param_dict_to_markdown	Format the *Parameters* as markdown string.

flatten_parameter_dict

glotaran.parameter.parameters.**flatten_parameter_dict**(*parameter_dict: dict*) → Generator[tuple[str, list[Any], dict[str, Any] | None], None, None]

Flatten a parameter dictionary.

Parameters
parameter_dict (*dict*) – The parameter dictionary.

Yields
tuple[str, list[Any], dict | None – The concatenated keys, the parameter definition and default options.

param_dict_to_markdown

glotaran.parameter.parameters.**param_dict_to_markdown**(*parameters: dict | list*, *float_format: str = '.3e'*, *depth: int = 0*, *label: str | None = None*) → *MarkdownStr*

Format the *Parameters* as markdown string.

This is done by recursing the nested *Parameters* tree.

Parameters

- **parameters** (*dict | list*) – The parameter dict or list.
- **float_format** (*str*) – Format string for floating point numbers, by default ".3e"
- **depth** (*int*) – The depth of the parameter dict.
- **label** (*str | None*) – The label of the parameter dict.

Returns
The markdown representation as string.

Return type
MarkdownStr

Classes

Summary

Parameters	A container for Parameter.

Parameters

class glotaran.parameter.parameters.**Parameters**(*parameters: dict[str, glotaran.parameter.parameter.Parameter]*)

Bases: object

A container for Parameter.

Create *Parameters*.

Parameters
parameters (*dict[str, Parameter]*) – A parameter list containing parameters

Returns
The created *Parameters*.

Return type
'Parameters'

Attributes Summary

labels	List of all labels.

labels

Parameters.**labels**

List of all labels.

Return type
list[str]

Methods Summary

all	Iterate over all parameters.
copy	Create a copy of the *Parameters*.
from_dataframe	Create a *Parameters* from a `pandas.DataFrame`.
from_dict	Create a *Parameters* from a dictionary.
from_list	Create *Parameters* from a list.
from_parameter_dict_list	Create *Parameters* from a list of parameter dictionaries.
get	Get a `Parameter` by its label.
get_label_value_and_bounds_arrays	Return a arrays of all parameter labels, values and bounds.
has	Check if a parameter with the given label is in the group or in a subgroup.
loader	Create a *Parameters* instance from the specs defined in a file.
markdown	Format the `ParameterGroup` as markdown string.
set_from_history	Update the *Parameters* with values from a parameter history.
set_from_label_and_value_arrays	Update the parameter values from a list of labels and values.
to_dataframe	Create a pandas data frame from the group.
to_parameter_dict_list	Create list of parameter dictionaries from the group.
to_parameter_dict_or_list	Convert to a dict or list of parameter definitions.
update_parameter_expression	Update all parameters which have an expression.

all

`Parameters.all()` → Generator[*Parameter*, None, None]

Iterate over all parameters.

Yields

Parameter – A parameter in the parameters.

copy

`Parameters.copy()` → *Parameters*

Create a copy of the *Parameters*.

Returns

- *Parameters* – A copy of the *Parameters*.
- **.. # noqa** (*D414*)

from_dataframe

classmethod Parameters.**from_dataframe**(*df: DataFrame*, *source: str = 'DataFrame'*) → *Parameters*

Create a *Parameters* from a pandas.DataFrame.

Parameters

- **df** (*pd.DataFrame*) – The source data frame.
- **source** (*str*) – Optional name of the source file, used for error messages.

Raises

ValueError – Raised if the columns 'label' or 'value' doesn't exist. Also raised if the columns 'minimum', 'maximum' or 'values' contain non numeric values or if the columns 'non-negative' or 'vary' are no boolean.

Returns

- *Parameters* – The created parameter group.
- **.. # noqa** (*D414*)

from_dict

classmethod Parameters.**from_dict**(*parameter_dict: dict[str, dict[str, Any] | list[float | list[Any]]]*) → *Parameters*

Create a *Parameters* from a dictionary.

Parameters

parameter_dict (*dict[str, dict[str, Any] | list[float | list[Any]]]*) – A parameter dictionary containing parameters.

Returns

- *Parameters* – The created *Parameters*
- **.. # noqa** (*D414*)

from_list

classmethod Parameters.**from_list**(*parameter_list: list[float | int | str | list[Any] | dict[str, Any]]*) → *Parameters*

Create *Parameters* from a list.

Parameters

parameter_list (*list[float | list[Any]]*) – A parameter list containing parameters

Returns

- *Parameters* – The created *Parameters*.
- **.. # noqa** (*D414*)

from_parameter_dict_list

classmethod Parameters.**from_parameter_dict_list**(*parameter_dict_list: list[dict[str, Any]]*) → *Parameters*

Create *Parameters* from a list of parameter dictionaries.

Parameters
parameter_dict_list (*list[dict[str, Any]]*) – A list of parameter dictionaries.

Returns

- *Parameters* – The created *Parameters*.
- **.. # noqa** (*D414*)

get

Parameters.**get**(*label: str*) → *Parameter*

Get a Parameter by its label.

Parameters
label (*str*) – The label of the parameter, with its path in a ParameterGroup prepended.

Returns
The parameter.

Return type
Parameter

Raises
ParameterNotFoundException – Raised if no parameter with the given label exists.

get_label_value_and_bounds_arrays

Parameters.**get_label_value_and_bounds_arrays**(*exclude_non_vary: bool = False*) → tuple[list[str], numpy.ndarray, numpy.ndarray, numpy.ndarray]

Return a arrays of all parameter labels, values and bounds.

Parameters
exclude_non_vary (*bool*) – If true, parameters with *vary=False* are excluded.

Returns
A tuple containing a list of parameter labels and an array of the values, lower and upper bounds.

Return type
tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

has

Parameters.**has**(*label: str*) → bool

Check if a parameter with the given label is in the group or in a subgroup.

Parameters
label (*str*) – The label of the parameter, with its path in a ParameterGroup prepended.

Returns
Whether a parameter with the given label exists in the group.

Return type
bool

loader

Parameters.**loader**(*format_name: str | None = None*, ***kwargs*) → *Parameters*

Create a *Parameters* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the load_parameters implementation of the project io plugin.

Returns

- *Parameters* – *Parameters* instance created from the file.
- **.. # noqa** (*D414*)

markdown

Parameters.**markdown**(*float_format: str = '.3e'*) → *MarkdownStr*

Format the ParameterGroup as markdown string.

This is done by recursing the nested ParameterGroup tree.

Parameters
float_format (*str*) – Format string for floating point numbers, by default ".3e"

Returns
The markdown representation as string.

Return type
MarkdownStr

set_from_history

Parameters.**set_from_history**(*history:* ParameterHistory, *index:* *int*)

Update the *Parameters* with values from a parameter history.

Parameters

- **history** (ParameterHistory) – The parameter history.
- **index** (*int*) – The history index.

set_from_label_and_value_arrays

Parameters.**set_from_label_and_value_arrays**(*labels:* *list[str]*, *values:* *ndarray*)

Update the parameter values from a list of labels and values.

Parameters

- **labels** (*list[str]*) – A list of parameter labels.
- **values** (*np.ndarray*) – An array of parameter values.

Raises

ValueError – Raised if the size of the labels does not match the stize of values.

to_dataframe

Parameters.**to_dataframe**() → DataFrame

Create a pandas data frame from the group.

Returns

The created data frame.

Return type

pd.DataFrame

to_parameter_dict_list

Parameters.**to_parameter_dict_list**() → list[dict[str, Any]]

Create list of parameter dictionaries from the group.

Returns

A list of parameter dictionaries.

Return type

list[dict[str, Any]]

to_parameter_dict_or_list

Parameters.**to_parameter_dict_or_list**(*serialize_parameters: bool = False*) → dict | list

Convert to a dict or list of parameter definitions.

Parameters

serialize_parameters (*bool*) – If true, the parameters will be serialized into list representation.

Returns

A dict or list of parameter definitions.

Return type

dict | list

update_parameter_expression

Parameters.**update_parameter_expression**()

Update all parameters which have an expression.

Raises

ValueError – Raised if an expression evaluates to a non-numeric value.

Methods Documentation

all() → Generator[*Parameter*, None, None]

Iterate over all parameters.

Yields

Parameter – A parameter in the parameters.

copy() → *Parameters*

Create a copy of the *Parameters*.

Returns

- *Parameters* – A copy of the *Parameters*.
- **.. # noqa** (*D414*)

classmethod from_dataframe(*df: DataFrame*, *source: str = 'DataFrame'*) → *Parameters*

Create a *Parameters* from a pandas.DataFrame.

Parameters

- **df** (*pd.DataFrame*) – The source data frame.
- **source** (*str*) – Optional name of the source file, used for error messages.

Raises

ValueError – Raised if the columns 'label' or 'value' doesn't exist. Also raised if the columns 'minimum', 'maximum' or 'values' contain non numeric values or if the columns 'non-negative' or 'vary' are no boolean.

Returns

- *Parameters* – The created parameter group.
- **.. # noqa** (*D414*)

classmethod from_dict(*parameter_dict: dict[str, dict[str, Any] | list[float | list[Any]]]*) → *Parameters*

Create a `Parameters` from a dictionary.

Parameters

parameter_dict (*dict[str, dict[str, Any] | list[float | list[Any]]]*) – A parameter dictionary containing parameters.

Returns

- *Parameters* – The created `Parameters`
- **.. # noqa** (*D414*)

classmethod from_list(*parameter_list: list[float | int | str | list[Any] | dict[str, Any]]*) → *Parameters*

Create `Parameters` from a list.

Parameters

parameter_list (*list[float | list[Any]]*) – A parameter list containing parameters

Returns

- *Parameters* – The created `Parameters`.
- **.. # noqa** (*D414*)

classmethod from_parameter_dict_list(*parameter_dict_list: list[dict[str, Any]]*) → *Parameters*

Create `Parameters` from a list of parameter dictionaries.

Parameters

parameter_dict_list (*list[dict[str, Any]]*) – A list of parameter dictionaries.

Returns

- *Parameters* – The created `Parameters`.
- **.. # noqa** (*D414*)

get(*label: str*) → *Parameter*

Get a `Parameter` by its label.

Parameters

label (*str*) – The label of the parameter, with its path in a `ParameterGroup` prepended.

Returns

The parameter.

Return type

Parameter

Raises

ParameterNotFoundException – Raised if no parameter with the given label exists.

get_label_value_and_bounds_arrays(*exclude_non_vary: bool = False*) → tuple[list[str], numpy.ndarray, numpy.ndarray, numpy.ndarray]

Return a arrays of all parameter labels, values and bounds.

Parameters
exclude_non_vary (*bool*) – If true, parameters with *vary=False* are excluded.

Returns
A tuple containing a list of parameter labels and an array of the values, lower and upper bounds.

Return type
tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

has(*label: str*) → bool

Check if a parameter with the given label is in the group or in a subgroup.

Parameters
label (*str*) – The label of the parameter, with its path in a ParameterGroup prepended.

Returns
Whether a parameter with the given label exists in the group.

Return type
bool

property labels: list[str]

List of all labels.

Return type
list[str]

loader(*format_name: str | None = None*, ***kwargs*) → *Parameters*

Create a *Parameters* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the load_parameters implementation of the project io plugin.

Returns

- *Parameters* – *Parameters* instance created from the file.
- **.. # noqa** (*D414*)

markdown(*float_format: str = '.3e'*) → *MarkdownStr*

Format the ParameterGroup as markdown string.

This is done by recursing the nested ParameterGroup tree.

Parameters
float_format (*str*) – Format string for floating point numbers, by default ".3e"

Returns
The markdown representation as string.

Return type
MarkdownStr

set_from_history(*history:* ParameterHistory, *index: int*)

Update the *Parameters* with values from a parameter history.

Parameters

- **history** (ParameterHistory) – The parameter history.
- **index** (*int*) – The history index.

set_from_label_and_value_arrays(*labels: list[str]*, *values: ndarray*)

Update the parameter values from a list of labels and values.

Parameters

- **labels** (*list[str]*) – A list of parameter labels.
- **values** (*np.ndarray*) – An array of parameter values.

Raises

ValueError – Raised if the size of the labels does not match the stize of values.

to_dataframe() → DataFrame

Create a pandas data frame from the group.

Returns

The created data frame.

Return type

pd.DataFrame

to_parameter_dict_list() → list[dict[str, Any]]

Create list of parameter dictionaries from the group.

Returns

A list of parameter dictionaries.

Return type

list[dict[str, Any]]

to_parameter_dict_or_list(*serialize_parameters: bool = False*) → dict | list

Convert to a dict or list of parameter definitions.

Parameters

serialize_parameters (*bool*) – If true, the parameters will be serialized into list representation.

Returns

A dict or list of parameter definitions.

Return type

dict | list

update_parameter_expression()

Update all parameters which have an expression.

Raises

ValueError – Raised if an expression evaluates to a non-numeric value.

Exceptions

Exception Summary

ParameterNotFoundException	Raised when a Parameter is not found.

ParameterNotFoundException

exception glotaran.parameter.parameters.**ParameterNotFoundException**(*label: str*)

Raised when a Parameter is not found.

16.1.9 plugin_system

Plugin system package containing all plugin related implementations.

Modules

glotaran.plugin_system.base_registry	Functionality to register, initialize and retrieve glotaran plugins.
glotaran.plugin_system.data_io_registration	Data Io registration convenience functions.
glotaran.plugin_system.io_plugin_utils	Utility functions for io plugin.
glotaran.plugin_system.megacomplex_registration	Megacomplex registration convenience functions.
glotaran.plugin_system.project_io_registration	Project Io registration convenience functions.

base_registry

Functionality to register, initialize and retrieve glotaran plugins.

Since this module is imported at the root `__init__.py` file all other glotaran imports should be used for typechecking only in the 'if TYPE_CHECKING' block. This is to prevent issues with circular imports.

Functions

Summary

add_instantiated_plugin_to_registry	Add instances of plugin_class to the given registry.
add_plugin_to_registry	Add a plugin with name `plugin_register_key` to the given registry.
full_plugin_name	Full name of a plugin instance/class similar to the `repr`.
get_method_from_plugin	Retrieve a method callabe from an class or instance plugin.
get_plugin_from_registry	Retrieve a plugin with name `plugin_register_key` is registered in a given registry.
is_registered_plugin	Check if a plugin with name `plugin_register_key` is registered in the given registry.
load_plugins	Initialize plugins registered under the entrypoint 'glotaran.plugins'.
methods_differ_from_baseclass	Check if a plugins methods implementation differ from its baseclass.
methods_differ_from_baseclass_table	Create table of which plugins methods differ from their baseclass.
registered_plugins	Names of the plugins in the given registry.
set_plugin	Set a plugins short name to a specific plugin referred by its full name.
show_method_help	Show help on a method as if it was called directly on it.
supported_file_extensions	Get file extensions for plugins that support all methods in `method_names`.

add_instantiated_plugin_to_registry

glotaran.plugin_system.base_registry.**add_instantiated_plugin_to_registry**(*plugin_register_keys: str | list[str], plugin_class: type[_PluginInstantiableType], plugin_registry: MutableMapping[str, _PluginInstantiableType], plugin_set_func_name: str*) → None

Add instances of plugin_class to the given registry.

Parameters

- **plugin_register_keys** (*str | list[str]*) – Name/-s of the plugin under which it is registered.
- **plugin_class** (*type[_PluginInstantiableType]*) – Pluginclass which should be instantiated with `plugin_register_keys` and added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginInstantiableType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.

See also:

`add_plugin_to_register`

add_plugin_to_registry

`glotaran.plugin_system.base_registry.`**add_plugin_to_registry**(*plugin_register_key: str*, *plugin: _PluginType*, *plugin_registry: MutableMapping[str, _PluginType]*, *plugin_set_func_name: str*, *instance_identifier: str = ''*) → None

Add a plugin with name `plugin_register_key` to the given registry.

In addition it also adds the plugin with it full import path name as key, which allows for a better reproducibility in case there are conflicting plugins.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin** (*_PluginType*) – Plugin to be added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- **instance_identifier** (*str*) – Used to differentiate between plugin instances (e.g. different format for IO plugins)

Raises

ValueError – If `plugin_register_key` has the character '.' in it.

See also:

`add_instantiated_plugin_to_register`, *full_plugin_name*

full_plugin_name

glotaran.plugin_system.base_registry.**full_plugin_name**(*plugin: object | type[object]*) → str

Full name of a plugin instance/class similar to the `repr`.

Parameters
plugin (*object | type[object]*) – plugin instance/class

Examples

```
>>> from glotaran.builtin.io.sdt.sdt_file_reader import SdtDataIo
>>> full_plugin_name(SdtDataIo)
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
>>> full_plugin_name(SdtDataIo("sdt"))
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
```

Returns
Full name of the plugin.

Return type
str

get_method_from_plugin

glotaran.plugin_system.base_registry.**get_method_from_plugin**(*plugin: object | type[object]*, *method_name: str*) → Callable[..., Any]

Retrieve a method callabe from an class or instance plugin.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_megacomplex.

Returns
Method callable.

Return type
Callable[..., Any]

Raises

- **ValueError** – If plugin has an attribute with that name but it isn't callable.
- **ValueError** – If plugin misses the attribute.

get_plugin_from_registry

glotaran.plugin_system.base_registry.**get_plugin_from_registry**(*plugin_register_key: str*, *plugin_registry: MutableMapping[str, _PluginType]*, *not_found_error_message: str*) → _PluginType

Retrieve a plugin with name `plugin_register_key` is registered in a given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **not_found_error_message** (*str*) – Error message to be shown if the plugin wasn't found.

Returns
Plugin from the plugin Registry.

Return type
_PluginType

Raises
ValueError – If there was no plugin registered under the name `plugin_register_key`.

is_registered_plugin

glotaran.plugin_system.base_registry.**is_registered_plugin**(*plugin_register_key: str*, *plugin_registry: MutableMapping[str, _PluginType]*) → bool

Check if a plugin with name `plugin_register_key` is registered in the given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.

Returns
Whether or not a plugin is in the registry.

Return type
bool

load_plugins

glotaran.plugin_system.base_registry.**load_plugins**()

Initialize plugins registered under the entrypoint 'glotaran.plugins'.

For an entry_point to be considered a glotaran plugin it just needs to start with 'glotaran.plugins', which allows for an easy extendability.

Currently used builtin entrypoints are:

- `glotaran.plugins.data_io`
- `glotaran.plugins.megacomplex`
- `glotaran.plugins.project_io`

methods_differ_from_baseclass

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass**(*method_names: str | Sequence[str]*, *plugin: GenericPluginInstance | type[GenericPluginInstance]*, *base_class: type[GenericPluginInstance]*) → Generator[bool, None, None]

Check if a plugins methods implementation differ from its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' list.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s
- **plugin** (*GenericPluginInstance | type[GenericPluginInstance]*) – Plugin class or instance.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.

Yields

bool – Whether or not a plugins method differs from the implementation in `base_class`.

methods_differ_from_baseclass_table

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass_table**(*method_names: str | Sequence[str], plugin_registry_keys: str | Sequence[str], get_plugin_function: Callable[[str], GenericPluginInstance | type[GenericPluginInstance]], base_class: type[GenericPluginInstance], plugin_names: bool = False*) → Generator[list[str | bool], None, None]

Create table of which plugins methods differ from their baseclass.

This uses the assumption that all plugins have the same `base_class`.

The main purpose of this function is to show the user which plugin implements which methods differently than its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' table.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s.
- **plugin_registry_keys** (*str | list[str]*) – Keys the plugins are registered under (e.g. return value of the implementation of func:*registered_plugins*)
- **get_plugin_function** (*Callable[[str], GenericPluginInstance | type[GenericPluginInstance]]*) – Function to get plugin from plugin registry.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.
- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the lists.

Yields

list[str | bool] – Row with the first value being the `plugin_registry_key` and the others whether or not a plugins method differs from `base_class`.

See also:

methods_differ_from_baseclass

registered_plugins

glotaran.plugin_system.base_registry.**registered_plugins**(*plugin_registry: MutableMapping[str, _PluginType]*, *full_names: bool = False*) → list[str]

Names of the plugins in the given registry.

Parameters

- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
List of plugin names in plugin_registry.

Return type
list[str]

set_plugin

glotaran.plugin_system.base_registry.**set_plugin**(*plugin_register_key: str*, *full_plugin_name: str*, *plugin_registry: MutableMapping[str, _PluginType]*, *plugin_register_key_name: str = 'format_name'*) → None

Set a plugins short name to a specific plugin referred by its full name.

This can be used to ensure that a specific plugin is used in case there are conflicting plugins installed.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be set in to.
- **plugin_register_key_name** (*str*) – Name of the arg passed `plugin_register_key` in the function that implements `set_plugin`.

Raises

- **ValueError** – If `plugin_register_key` has the character '.' in it.
- **ValueError** – If there isn't a registered plugin with the key `full_plugin_name`.

See also:

add_plugin_to_registry, *full_plugin_name*

show_method_help

glotaran.plugin_system.base_registry.**show_method_help**(*plugin: object | type[object]*, *method_name: str*) → None

Show help on a method as if it was called directly on it.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_megacomplex.

supported_file_extensions

glotaran.plugin_system.base_registry.**supported_file_extensions**(*method_names: str | Sequence[str]*, *plugin_registry_keys: str | Sequence[str]*, *get_plugin_function: Callable[[str], GenericPluginInstance | type[GenericPluginInstance]]*, *base_class: type[GenericPluginInstance]*) → Generator[str, None, None]

Get file extensions for plugins that support all methods in `method_names`.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s.
- **plugin_registry_keys** (*str | list[str]*) – Keys the plugins are registered under (e.g. return value of the implementation of func:*registered_plugins*)
- **get_plugin_function** (*Callable[[str], GenericPluginInstance | type[GenericPluginInstance]]*) – Function to get plugin from plugin registry.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.

Yields

Generator[str, None, None] – File extension supported by all methods in `method_names`.

See also:

methods_differ_from_baseclass, *methods_differ_from_baseclass_table*

Exceptions

Exception Summary

`PluginOverwriteWarning`	Warning used if a plugin tries to overwrite and existing plugin.

PluginOverwriteWarning

exception glotaran.plugin_system.base_registry.**PluginOverwriteWarning**(**args: Any*, *old_key: str*, *old_plugin: object | type[object]*, *new_plugin: object | type[object]*, *plugin_set_func_name: str*)

Warning used if a plugin tries to overwrite and existing plugin.

Use old and new plugin and keys to give verbose warning message.

Parameters

- **old_key** (*str*) – Old registry key.
- **old_plugin** (*object | type[object]*) – Old plugin ('registry[old_key]').
- **new_plugin** (*object | type[object]*) – New Plugin ('registry[new_key]').
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- ***args** (*Any*) – Additional args passed to the super constructor.

data_io_registration

Data Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

data_io_plugin_table	Return registered data io plugins and which functions they support as markdown table.
get_data_io	Retrieve a data io plugin from the data_io registry.
get_dataloader	Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.
get_datasaver	Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.
is_known_data_format	Check if a data format is in the data_io registry.
known_data_formats	Names of the registered data io plugins.
load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray.
register_data_io	Register data io plugins to one or more formats.
save_dataset	Save data from xarray.Dataset or xarray.DataArray to a file.
set_data_plugin	Set the plugin used for a specific data format.
show_data_io_method_help	Show help for the implementation of data io plugin methods.
supported_file_extensions_data_io	Get data io formats that support all methods in `method_names`.

data_io_plugin_table

glotaran.plugin_system.data_io_registration.**data_io_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *MarkdownStr*

Return registered data io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns

Markdown table of data io plugins.

Return type

MarkdownStr

get_data_io

glotaran.plugin_system.data_io_registration.**get_data_io**(*format_name: str*) → *DataIoInterface*

Retrieve a data io plugin from the data_io registry.

Parameters
format_name (*str*) – Name of the data io plugin under which it is registered.

Returns
Data io plugin instance.

Return type
DataIoInterface

get_dataloader

glotaran.plugin_system.data_io_registration.**get_dataloader**(*format_name: str*) → DataLoader

Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters
format_name (*str*) – Format the dataloader should be able to read.

Returns
Function to load data of format `format_name` as xarray.Dataset or xarray.DataArray.

Return type
DataLoader

get_datasaver

glotaran.plugin_system.data_io_registration.**get_datasaver**(*format_name: str*) → DataSaver

Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters
format_name (*str*) – Format the datawriter should be able to write.

Returns
Function to write xarray.Dataset to the format `format_name` .

Return type
DataSaver

is_known_data_format

glotaran.plugin_system.data_io_registration.**is_known_data_format**(*format_name: str*) → bool

Check if a data format is in the data_io registry.

Parameters
format_name (*str*) – Name of the data io plugin under which it is registered.

Returns
Whether or not the data format is a registered data io plugins.

Return type
bool

known_data_formats

glotaran.plugin_system.data_io_registration.**known_data_formats**(*full_names: bool = False*) → list[str]

Names of the registered data io plugins.

Parameters
full_names (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
List of registered data io plugins.

Return type
list[str]

load_dataset

glotaran.plugin_system.data_io_registration.**load_dataset**(*file_name: StrOrPath*, *format_name: str | None = None*, ***kwargs: Any*) → xr.Dataset

Read data from a file to xarray.Dataset or xarray.DataArray.

Parameters

- **file_name** (*StrOrPath*) – File containing the data.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `read_dataset` implementation of the data io plugin. If you aren't sure about those use `get_dataloader` to get the implementation with the proper help and autocomplete.

Returns
Data loaded from the file.

Return type
xr.Dataset

register_data_io

glotaran.plugin_system.data_io_registration.**register_data_io**(*format_names: str | list[str]*) → Callable[[type[*DataIoInterface*]], type[*DataIoInterface*]]

Register data io plugins to one or more formats.

Decorate a data io plugin class with `@register_data_io(format_name|[*format_names])` to add it to the registry.

Parameters
format_names (*str | list[str]*) – Name of the data io plugin under which it is registered.

Returns
Inner decorator function.

Return type
Callable[[type[*DataIoInterface*]], type[*DataIoInterface*]]

Examples

```
>>> @register_data_io("my_format_1")
... class MyDataIo1(DataIoInterface):
...     pass
```

```
>>> @register_data_io(["my_format_1", "my_format_1_alias"])
... class MyDataIo2(DataIoInterface):
...     pass
```

save_dataset

glotaran.plugin_system.data_io_registration.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: StrOrPath*, *format_name: str | None = None*, ***, *data_filters: list[str] | None = None*, *allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → None

Save data from xarray.Dataset or xarray.DataArray to a file.

Parameters

- **dataset** (*xr.Dataset | xr.DataArray*) – Data to be written to file.
- **file_name** (*StrOrPath*) – File to write the data to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **data_filters** (*list[str] | None*) – Optional list of items in the dataset to be saved.

- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `file_name` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `write_dataset` implementation of the data io plugin. If you aren't sure about those use `get_datasaver` to get the implementation with the proper help and autocomplete.

set_data_plugin

glotaran.plugin_system.data_io_registration.**set_data_plugin**(*format_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific data format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_dataset()*
- *save_dataset()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_data_io_method_help

glotaran.plugin_system.data_io_registration.**show_data_io_method_help**(*format_name: str*, *method_name: Literal['load_dataset', 'save_dataset']*) → None

Show help for the implementation of data io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_dataset', 'save_dataset'}*) – Method name

supported_file_extensions_data_io

glotaran.plugin_system.data_io_registration.**supported_file_extensions_data_io**(*method_names: str | Sequence[str]*) → Generator[str, None, None]

Get data io formats that support all methods in `method_names`.

Parameters
method_names (*str | Sequence[str]*) – Names of Methods that need to support the file extension.

Yields
Generator[str, None, None] – File extension supported by all methods in `method_names`.

See also:

`supported_file_extensions`, `DATA_IO_METHODS`

io_plugin_utils

Utility functions for io plugin.

Functions

Summary

bool_str_repr	Replace boolean value with string repr.
bool_table_repr	Replace boolean value with string repr for all table values.
infer_file_format	Infer format of a file if it exists.
not_implemented_to_value_error	Decorate a function to raise ValueError instead of NotImplementedError.
protect_from_overwrite	Raise FileExistsError if files already exists and allow_overwrite isn't True.

bool_str_repr

glotaran.plugin_system.io_plugin_utils.**bool_str_repr**(*value: Any*, *true_repr: str = '*'*, *false_repr: str = '/'*) → Any

Replace boolean value with string repr.

This function is a helper for table representation (e.g. with tabulate) of boolean values.

Parameters

- **value** (*Any*) – Arbitrary value
- **true_repr** (*str*) – Desired repr for `True`, by default "*"
- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns
Original value or desired repr for bool

Return type
Any

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(map(lambda x: map(bool_table_repr, x), table_data)))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

bool_table_repr

glotaran.plugin_system.io_plugin_utils.**bool_table_repr**(*table_data: Iterable[Iterable[Any]]*, *true_repr: str = '*'*, *false_repr: str = '/'*) → Iterator[Iterator[Any]]

Replace boolean value with string repr for all table values.

This function is an implementation of *bool_str_repr()* for a 2D table, for easy usage with tabulate.

Parameters

- **table_data** (*Iterable[Iterable[Any]]*) – Data of the table e.g. a list of lists.
- **true_repr** (*str*) – Desired repr for `True`, by default "*"
- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns
`table_data` with original values or desired repr for bool

Return type
Iterator[Iterator[Any]]

See also:

bool_str_repr

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(bool_table_repr(table_data))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

infer_file_format

glotaran.plugin_system.io_plugin_utils.**infer_file_format**(*file_path: StrOrPath*, ***, *needs_to_exist: bool = True*, *allow_folder=False*) → str

Infer format of a file if it exists.

Parameters

- **file_path** (*StrOrPath*) – Path/str to the file.
- **needs_to_exist** (*bool*) – Whether or not a file need to exists for an successful format inferring. While write functions don't need the file to exists, load functions do.
- **allow_folder** (*bool*) – Whether or not to allow the format to be `folder`. This is only used in `save_result`.

Returns
File extension without the leading dot.

Return type
str

Raises

- **ValueError** – If file doesn't exists.
- **ValueError** – If file has no extension.

not_implemented_to_value_error

glotaran.plugin_system.io_plugin_utils.**not_implemented_to_value_error**(*func: DecoratedFunc*) → DecoratedFunc

Decorate a function to raise ValueError instead of NotImplementedError.

This decorator is supposed to be used on functions which call functions that might raise a NotImplementedError, but raise ValueError instead with the same error text.

Parameters
func (*DecoratedFunc*) – Function to be decorated.

Returns
Wrapped function.

Return type
DecoratedFunc

protect_from_overwrite

glotaran.plugin_system.io_plugin_utils.**protect_from_overwrite**(*path: str | PathLike[str]*, *, *allow_overwrite: bool = False*) → None

Raise FileExistsError if files already exists and allow_overwrite isn't True.

As a side effect this also creates the parent directory of a file if it does not exist.

Parameters

- **path** (*str*) – Path to a file or folder.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False

Raises

- **FileExistsError** – If path points to an existing file.
- **FileExistsError** – If path points to an existing folder which is not empty.

megacomplex_registration

Megacomplex registration convenience functions.

Functions

Summary

get_megacomplex	Retrieve a megacomplex from the megacomplex registry.
is_known_megacomplex	Check if a megacomplex is in the megacomplex registry.
known_megacomplex_names	Names of the registered megacomplexs.
megacomplex_plugin_table	Return registered megacomplex plugins as markdown table.
register_megacomplex	Add a megacomplex to the megacomplex registry.
set_megacomplex_plugin	Set the plugin used for a specific megacomplex name.

get_megacomplex

glotaran.plugin_system.megacomplex_registration.**get_megacomplex**(*megacomplex_type: str*) → type[Megacomplex]

Retrieve a megacomplex from the megacomplex registry.

Parameters
megacomplex_type (*str*) – Name of the megacomplex under which it is registered.

Returns
Megacomplex class

Return type
type[Megacomplex]

is_known_megacomplex

glotaran.plugin_system.megacomplex_registration.**is_known_megacomplex**(*megacomplex_type: str*) → bool

Check if a megacomplex is in the megacomplex registry.

Parameters
megacomplex_type (*str*) – Name of the megacomplex under which it is registered.

Returns
Whether or not the megacomplex is registered.

Return type
bool

known_megacomplex_names

glotaran.plugin_system.megacomplex_registration.**known_megacomplex_names**(*full_names: bool = False*) → list[str]

Names of the registered megacomplexs.

Parameters
full_names (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
List of registered megacomplexs.

Return type
list[str]

megacomplex_plugin_table

glotaran.plugin_system.megacomplex_registration.**megacomplex_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *MarkdownStr*

Return registered megacomplex plugins as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
Markdown table of megacomplexnames.

Return type
MarkdownStr

register_megacomplex

glotaran.plugin_system.megacomplex_registration.**register_megacomplex**(*megacomplex_type: str*, *megacomplex: type[Megacomplex]*) → None

Add a megacomplex to the megacomplex registry.

Parameters

- **megacomplex_type** (*str*) – Name of the megacomplex under which it is registered.
- **megacomplex** (*type[Megacomplex]*) – megacomplex class to be registered.

set_megacomplex_plugin

glotaran.plugin_system.megacomplex_registration.**set_megacomplex_plugin**(*megacomplex_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific megacomplex name.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific megacomplex name.

Effected functions:

- `optimize()`

Parameters

- **megacomplex_name** (*str*) – Name of the megacomplex to use the plugin for.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

project_io_registration

Project Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

get_project_io	Retrieve a data io plugin from the project_io registry.
get_project_io_method	Retrieve implementation of project io functionality for the format 'format_name'.
is_known_project_format	Check if a data format is in the project_io registry.
known_project_formats	Names of the registered project io plugins.
load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a `Parameters` instance from the specs defined in a file.
load_result	Create a `Result` instance from the specs defined in a file.
load_scheme	Create a `Scheme` instance from the specs defined in a file.
project_io_plugin_table	Return registered project io plugins and which functions they support as markdown table.
register_project_io	Register project io plugins to one or more formats.
save_model	Save a `Model` instance to a spec file.
save_parameters	Save a `Parameters` instance to a spec file.
save_result	Write a `Result` instance to a spec file.
save_scheme	Save a `Scheme` instance to a spec file.
set_project_plugin	Set the plugin used for a specific project format.
show_project_io_method_help	Show help for the implementation of project io plugin methods.
supported_file_extensions_project_io	Get project io formats that support all methods in `method_names`.

get_project_io

glotaran.plugin_system.project_io_registration.**get_project_io**(*format_name: str*) → *ProjectIoInterface*

Retrieve a data io plugin from the project_io registry.

Parameters
format_name (*str*) – Name of the data io plugin under which it is registered.

Returns
Project io plugin instance.

Return type
ProjectIoInterface

get_project_io_method

glotaran.plugin_system.project_io_registration.**get_project_io_method**(*format_name: str*, *method_name: ProjectIoMethods*) → Callable[..., Any]

Retrieve implementation of project io functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters

- **format_name** (*str*) – Format the dataloader should be able to read.
- **method_name** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name, e.g. load_model.

Returns
The function which is called in the background by the convenience functions.

Return type
Callable[..., Any]

is_known_project_format

glotaran.plugin_system.project_io_registration.**is_known_project_format**(*format_name: str*) → bool

Check if a data format is in the project_io registry.

Parameters
format_name (*str*) – Name of the project io plugin under which it is registered.

Returns
Whether or not the data format is a registered project io plugin.

Return type
bool

known_project_formats

glotaran.plugin_system.project_io_registration.**known_project_formats**(*full_names: bool = False*) → list[str]

Names of the registered project io plugins.

Parameters
full_names (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
List of registered project io plugins.

Return type
list[str]

load_model

glotaran.plugin_system.project_io_registration.**load_model**(*file_name: StrOrPath*, *format_name: str | None = None*, ***kwargs: Any*) → *Model*

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns
Model instance created from the file.

Return type
Model

load_parameters

glotaran.plugin_system.project_io_registration.**load_parameters**(*file_name: StrOrPath*, *format_name: str | None = None*, ***kwargs*) → *Parameters*

Create a `Parameters` instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_parameters` implementation of the project io plugin.

Returns

- *Parameters* – `Parameters` instance created from the file.
- **.. # noqa** (*D414*)

load_result

glotaran.plugin_system.project_io_registration.**load_result**(*result_path: StrOrPath*, *format_name: str | None = None*, ***kwargs: Any*) → *Result*

Create a `Result` instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str | None*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns
`Result` instance created from the saved format.

Return type
Result

load_scheme

glotaran.plugin_system.project_io_registration.**load_scheme**(*file_name: StrOrPath*, *format_name: str | None = None*, ***kwargs: Any*) → *Scheme*

Create a `Scheme` instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns
`Scheme` instance created from the file.

Return type
Scheme

project_io_plugin_table

glotaran.plugin_system.project_io_registration.**project_io_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *MarkdownStr*

Return registered project io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns
Markdown table of project io plugins.

Return type
MarkdownStr

register_project_io

glotaran.plugin_system.project_io_registration.**register_project_io**(*format_names: str | list[str]*) → Callable[[type[*ProjectIoInterface*]], type[*ProjectIoInterface*]]

Register project io plugins to one or more formats.

Decorate a project io plugin class with `@register_project_io(format_name|[*format_names])` to add it to the registry.

Parameters
format_names (*str | list[str]*) – Name of the project io plugin under which it is registered.

Returns
Inner decorator function.

Return type
Callable[[type[*ProjectIoInterface*]], type[*ProjectIoInterface*]]

Examples

```
>>> @register_project_io("my_format_1")
... class MyProjectIo1(ProjectIoInterface):
...     pass
```

```
>>> @register_project_io(["my_format_1", "my_format_1_alias"])
... class MyProjectIo2(ProjectIoInterface):
...     pass
```

save_model

glotaran.plugin_system.project_io_registration.**save_model**(*model:* Model, *file_name:* *StrOrPath*, *format_name:* *str* | *None* = *None*, ***, *allow_overwrite:* *bool* = *False*, *update_source_path:* *bool* = *True*, ***kwargs:* *Any*) → None

Save a `Model` instance to a spec file.

Parameters

- **model** (`Model`) – `Model` instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the model specs to.
- **format_name** (*str | None*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `file_name` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_model` implementation of the project io plugin.

save_parameters

glotaran.plugin_system.project_io_registration.**save_parameters**(*parameters:* Parameters, *file_name: StrOrPath*, *format_name: str* | *None* = *None*, ***, *allow_overwrite:* *bool* = *False*, *update_source_path:* *bool* = *True*, ***kwargs: Any*) → None

Save a `Parameters` instance to a spec file.

Parameters

- **parameters** (Parameters) – Parameters instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the parameter specs to.
- **format_name** (*str | None*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the source_path attribute to file_name when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the save_parameters implementation of the project io plugin.

save_result

glotaran.plugin_system.project_io_registration.**save_result**(*result: Result*, *result_path: StrOrPath*, *format_name: str | None = None*, ***, *allow_overwrite: bool = False*, *update_source_path: bool = True*, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*, ***kwargs: Any*) → list[str]

Write a Result instance to a spec file.

Parameters

- **result** (Result) – Result instance to write.
- **result_path** (*StrOrPath*) – Path to write the result data to.
- **format_name** (*str | None*) – Format the result should be saved in, if not provided and it is a file it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the source_path attribute to result_path when saving. by default True
- **saving_options** (SavingOptions) – Options for the saved result.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the save_result implementation of the project io plugin.

Returns

List of file paths which were saved.

Return type

list[str] | None

save_scheme

glotaran.plugin_system.project_io_registration.**save_scheme**(*scheme:* Scheme, *file_name: StrOrPath*, *format_name: str | None = None*, *, *allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → None

Save a Scheme instance to a spec file.

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the scheme specs to.
- **format_name** (*str | None*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the source_path attribute to file_name when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the save_scheme implementation of the project io plugin.

set_project_plugin

glotaran.plugin_system.project_io_registration.**set_project_plugin**(*format_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific project format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_model()*
- *save_model()*
- *load_parameters()*
- *save_parameters()*
- *load_scheme()*
- *save_scheme()*
- *load_result()*
- *save_result()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_project_io_method_help

glotaran.plugin_system.project_io_registration.**show_project_io_method_help**(*format_name: str*, *method_name: ProjectIoMethods*) → None

Show help for the implementation of project io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name.

supported_file_extensions_project_io

glotaran.plugin_system.project_io_registration.**supported_file_extensions_project_io**(*method_names: str | Sequence[str]*) → Generator[str, None, None]

Get project io formats that support all methods in `method_names`.

Parameters

method_names (*str | Sequence[str]*) – Names of Methods that need to support the file extension.

Yields

Generator[str, None, None] – File extension supported by all methods in `method_names`.

See also:

`supported_file_extensions`, `PROJECT_IO_METHODS`

16.1.10 project

The glotaran project package.

Modules

glotaran.project.dataclass_helpers	Contains helper methods for dataclasses.
glotaran.project.generators	The glotaran generator package.
glotaran.project.project	The glotaran project module.
glotaran.project.project_data_registry	The glotaran data registry module.
glotaran.project.project_model_registry	The glotaran model registry module.
glotaran.project.project_parameter_registry	The glotaran parameter registry module.
glotaran.project.project_registry	The glotaran registry module.
glotaran.project.project_result_registry	The glotaran result registry module.
glotaran.project.result	The result class for global analysis.
glotaran.project.scheme	The module for :class:`Scheme`.

dataclass_helpers

Contains helper methods for dataclasses.

Functions

Summary

asdict	Create a dictionary containing all fields of the dataclass.
exclude_from_dict_field	Create a dataclass field with which will be excluded from `asdict`.
file_loadable_field	Create a dataclass field which can be and object of type `targetClass` or file path.
file_loader_factory	Create `file_loader` functions to load `targetClass` from file.
fromdict	Create a dataclass instance from a dict and loads all file represented fields.
init_file_loadable_fields	Load objects into class when dataclass is initialized with paths.

asdict

glotaran.project.dataclass_helpers.**asdict**(*dataclass: DataclassInstance*, *folder: Path | None = None*) → dict[str, Any]

Create a dictionary containing all fields of the dataclass.

Parameters

- **dataclass** (*DataclassInstance*) – A dataclass instance.

- **folder** (*Path | None*) – Parent folder of `FileLoadable` fields. by default None

Returns
The dataclass represented as a dictionary.

Return type
dict[str, Any]

exclude_from_dict_field

glotaran.project.dataclass_helpers.**exclude_from_dict_field**(*default: DefaultType = <dataclasses._MISSING_TYPE object>*) → DefaultType

Create a dataclass field with which will be excluded from `asdict`.

Parameters
default (*DefaultType*) – The default value of the field.

Returns
The created field.

Return type
DefaultType

file_loadable_field

glotaran.project.dataclass_helpers.**file_loadable_field**(*targetClass: type[FileLoadable]*, *, *is_wrapper_class=False*) → FileLoadable

Create a dataclass field which can be and object of type `targetClass` or file path.

Parameters

- **targetClass** (*type[FileLoadable]*) – Class the resulting value should be an instance of.
- **is_wrapper_class** (*bool*) – Whether or not `targetClass` is a wrapper class, so the isinstance check will be ignored and instead the responsibility for supported types lies at the implementation of the loader.

Notes

This also requires to add `init_file_loadable_fields` in the `__post_init__` method.

Returns
Instance of `targetClass`.

Return type
FileLoadable

See also:

init_file_loadable_fields

file_loader_factory

glotaran.project.dataclass_helpers.**file_loader_factory**(*targetClass: type[FileLoadable]*, ***, *is_wrapper_class: bool = False*) → Callable[[FileLoadable | str | Path], FileLoadable]

Create `file_loader` functions to load `targetClass` from file.

Parameters

- **targetClass** (*type[FileLoadable]*) – Class the loader function should return an instance of.
- **is_wrapper_class** (*bool*) – Whether or not `targetClass` is a wrapper class, so the isinstance check will be ignored and instead the responsibility for supported types lies at the implementation of the loader.

Returns

file_loader – Function to load `FileLoadable` from source file or return instance if already loaded.

Return type

Callable[[FileLoadable | str | Path], FileLoadable]

fromdict

glotaran.project.dataclass_helpers.**fromdict**(*dataclass_type: type[DataclassInstanceType]*, *dataclass_dict: dict[str, Any]*, *folder: Path | None = None*) → DataclassInstanceType

Create a dataclass instance from a dict and loads all file represented fields.

Parameters

- **dataclass_type** (*type[DataclassInstanceType]*) – A dataclass type.
- **dataclass_dict** (*dict[str, Any]*) – A dict for instancing the the dataclass.
- **folder** (*Path*) – The root folder for file paths. If `None` file paths are consider absolute.

Returns

Created instance of dataclass_type.

Return type

DataclassInstanceType

init_file_loadable_fields

glotaran.project.dataclass_helpers.**init_file_loadable_fields**(*dataclass_instance: DataclassInstance*)

Load objects into class when dataclass is initialized with paths.

If the class has file_loadable fields, this needs be called in the `__post_init__` method of that class.

Parameters

dataclass_instance (*DataclassInstance*) – Instance of the dataclass being initialized. When used inside of `__post_init__` for the class itself use `self`.

See also:

file_loadable_field

generators

The glotaran generator package.

Modules

glotaran.project.generators.generator	The glotaran generator module.

generator

The glotaran generator module.

Functions

Summary

generate_model	Generate a model.
generate_model_yml	Generate a model as yml string.
generate_parallel_decay_model	Generate a parallel decay model dictionary.
generate_parallel_spectral_decay_model	Generate a parallel spectral decay model dictionary.
generate_sequential_decay_model	Generate a sequential decay model dictionary.
generate_sequential_spectral_decay_mode	Generate a sequential spectral decay model dictionary.

generate_model

glotaran.project.generators.generator.**generate_model**(*, *generator_name: str*, *generator_arguments:* GeneratorArguments) → *Model*

Generate a model.

Parameters

- **generator_name** (*str*) – The generator to use.
- **generator_arguments** (GeneratorArguments) – Arguments for the generator.

Returns
The generated model

Return type
Model

See also:

generate_parallel_decay_model, *generate_parallel_spectral_decay_model*, *generate_sequential_decay_model*, *generate_sequential_spectral_decay_model*

Raises
ValueError – Raised when an unknown generator is specified.

generate_model_yml

glotaran.project.generators.generator.**generate_model_yml**(*, *generator_name: str*, *generator_arguments:* GeneratorArguments) → str

Generate a model as yml string.

Parameters

- **generator_name** (*str*) – The generator to use.
- **generator_arguments** (GeneratorArguments) – Arguments for the generator.

Returns
The generated model yml string.

Return type
str

See also:

generate_parallel_decay_model, *generate_parallel_spectral_decay_model*, *generate_sequential_decay_model*, *generate_sequential_spectral_decay_model*

Raises
ValueError – Raised when an unknown generator is specified.

generate_parallel_decay_model

glotaran.project.generators.generator.**generate_parallel_decay_model**(*, *nr_compartments: int = 1*, *irf: bool = False*) → dict[str, Any]

Generate a parallel decay model dictionary.

Parameters

- **nr_compartments** (*int*) – The number of compartments.
- **irf** (*bool*) – Whether to add a gaussian irf.

Returns
The generated model dictionary.

Return type
dict[str, Any]

generate_parallel_spectral_decay_model

glotaran.project.generators.generator.**generate_parallel_spectral_decay_model**(*, *nr_compartments: int = 1*, *irf: bool = False*) → dict[str, Any]

Generate a parallel spectral decay model dictionary.

Parameters

- **nr_compartments** (*int*) – The number of compartments.
- **irf** (*bool*) – Whether to add a gaussian irf.

Returns
The generated model dictionary.

Return type
dict[str, Any]

generate_sequential_decay_model

glotaran.project.generators.generator.**generate_sequential_decay_model**(*nr_compartments: int = 1*, *irf: bool = False*) → dict[str, Any]

Generate a sequential decay model dictionary.

Parameters

- **nr_compartments** (*int*) – The number of compartments.
- **irf** (*bool*) – Whether to add a gaussian irf.

Returns
The generated model dictionary.

Return type
dict[str, Any]

generate_sequential_spectral_decay_model

glotaran.project.generators.generator.**generate_sequential_spectral_decay_model**(*, *nr_compartments: int = 1*, *irf: bool = False*) → dict[str, Any]

Generate a sequential spectral decay model dictionary.

Parameters

- **nr_compartments** (*int*) – The number of compartments.
- **irf** (*bool*) – Whether to add a gaussian irf.

Returns
The generated model dictionary.

Return type
dict[str, Any]

Classes

Summary

GeneratorArguments	Arguments used by `generate_model` and `generate_model`.

GeneratorArguments

class glotaran.project.generators.generator.**GeneratorArguments**

Bases: TypedDict

Arguments used by `generate_model` and `generate_model`.

Parameters

- **nr_compartments** (*int*) – The number of compartments.
- **irf** (*bool*) – Whether to add a gaussian irf.

See also:

generate_model, *generate_model_yml*

Attributes Summary

nr_compartments
irf

nr_compartments

GeneratorArguments.**nr_compartments: int**

irf

GeneratorArguments.**irf: bool**

Methods Summary

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If the key is not found, return the default if given; otherwise, raise a KeyError.
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

clear

GeneratorArguments.**clear**() → None. Remove all items from D.

copy

GeneratorArguments.**copy**() → a shallow copy of D

fromkeys

GeneratorArguments.**fromkeys**(*iterable*, *value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get

GeneratorArguments.**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items

GeneratorArguments.**items**() → a set-like object providing a view on D's items

keys

GeneratorArguments.**keys**() → a set-like object providing a view on D's keys

pop

GeneratorArguments.**pop**(*key*, *default=<unrepresentable>*, */*)

If the key is not found, return the default if given; otherwise, raise a KeyError.

popitem

GeneratorArguments.**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.

setdefault

GeneratorArguments.**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.

update

GeneratorArguments.**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]

values

GeneratorArguments.**values**() → an object providing a view on D's values

Methods Documentation

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*iterable*, *value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

irf: bool

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

nr_compartments: int

pop(*key*, *default=<unrepresentable>*, /)
If the key is not found, return the default if given; otherwise, raise a KeyError.

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.

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.

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]

values() → an object providing a view on D's values

project

The glotaran project module.

Classes

Summary

Project	A project represents a projectfolder on disk which contains a project file.

Project

class glotaran.project.project.**Project**(*file: Path*, *folder: Path | None = None*)

Bases: object

A project represents a projectfolder on disk which contains a project file.

A project file is a file in *yml* format with name *project.gta*

Attributes Summary

data	Get all project datasets.
folder	
has_data	Check if the project has datasets.
has_models	Check if the project has models.
has_parameters	Check if the project has parameters.
has_results	Check if the project has results.
models	Get all project models.
parameters	Get all project parameters.
results	Get all project results.
version	
file	

data

Project.**data**

Get all project datasets.

Returns
The models of the datasets.

Return type
Mapping[str, Path]

folder

```
Project.folder: Path = None
```

has_data

```
Project.has_data
```

Check if the project has datasets.

Returns
Whether the project has datasets.

Return type
bool

has_models

```
Project.has_models
```

Check if the project has models.

Returns
Whether the project has models.

Return type
bool

has_parameters

```
Project.has_parameters
```

Check if the project has parameters.

Returns
Whether the project has parameters.

Return type
bool

has_results

```
Project.has_results
```

Check if the project has results.

Returns
Whether the project has results.

Return type
bool

models

Project.**models**

Get all project models.

Returns
The models of the project.

Return type
Mapping[str, Path]

parameters

Project.**parameters**

Get all project parameters.

Returns
The parameters of the project.

Return type
Mapping[str, Path]

results

Project.**results**

Get all project results.

Returns
The results of the project.

Return type
Mapping[str, Path]

version

Project.**version: str**

file

Project.**file: Path**

Methods Summary

create	Create a new project folder and file.
create_scheme	Create a scheme for optimization.
generate_model	Generate a model.
generate_parameters	Generate parameters for a model.
get_latest_result_path	Get the path to a result with name `name`.
get_models_directory	Get the path to the model directory of the project.
get_parameters_directory	Get the path to the parameter directory of the project.
get_result_path	Get the path to a result with name `name`.
import_data	Import a dataset by saving it as an .nc file in the project's data folder.
load_data	Load a dataset, with SVD data if `add_svd` is `True`.
load_latest_result	Load a result.
load_model	Load a model.
load_parameters	Load parameters.
load_result	Load a result.
markdown	Format the project as a markdown text.
open	Open a new project.
optimize	Optimize a model.
show_model_definition	Show model definition file content with syntax highlighting.
show_parameters_definition	Show parameters definition file content with syntax highlighting.
validate	Check that the model is valid, list all issues in the model if there are any.

create

static Project.**create**(*folder: str | Path*, *allow_overwrite: bool = False*) → *Project*

Create a new project folder and file.

Parameters

- **folder** (*str | Path | None*) – The folder where the project will be created. If `None`, the current work directory will be used.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing project file.

Returns

The created project.

Return type

Project

Raises

FileExistsError – Raised if the project file already exists and *allow_overwrite=False*.

create_scheme

Project.**create_scheme**(*model_name: str*, *parameters_name: str*, *maximum_number_function_evaluations: int | None = None*, *clp_link_tolerance: float = 0.0*, *data_lookup_override: Mapping[str, LoadableDataset] | None = None*) → *Scheme*

Create a scheme for optimization.

Parameters

- **model_name** (*str*) – The model to optimize.
- **parameters_name** (*str*) – The initial parameters.
- **maximum_number_function_evaluations** (*int | None*) – The maximum number of function evaluations.
- **clp_link_tolerance** (*float*) – The CLP link tolerance.
- **data_lookup_override** (*Mapping[str, LoadableDataset] | None*) – Allows to bypass the default dataset lookup in the project `data` folder and use a different dataset for the optimization without changing the model. This is especially useful when working with preprocessed data. Defaults to `None`.

Returns
The created scheme.

Return type
Scheme

generate_model

Project.**generate_model**(*model_name: str*, *generator_name: str*, *generator_arguments: dict[str, Any]*, ***, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate a model.

Parameters

- **model_name** (*str*) – The name of the model.
- **generator_name** (*str*) – The generator for the model.
- **generator_arguments** (*dict[str, Any]*) – Arguments for the generator.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing model.
- **ignore_existing** (*bool*) – Whether to ignore generation of a model file if it already exists.

generate_parameters

Project.**generate_parameters**(*model_name: str*, *parameters_name: str | None = None*, ***, *format_name: Literal['yml', 'yaml', 'csv'] = 'csv'*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate parameters for a model.

Parameters

- **model_name** (*str*) – The model.
- **parameters_name** (*str | None*) – The name of the parameters. If None it will be <model_name>_parameters.
- **format_name** (*Literal["yml", "yaml", "csv"]*) – The parameter format.
- **allow_overwrite** (*bool*) – Whether to overwrite existing parameters.
- **ignore_existing** (*bool*) – Whether to ignore generation of a parameter file if it already exists.

get_latest_result_path

Project.**get_latest_result_path**(*result_name: str*) → Path

Get the path to a result with name name.

Parameters
result_name (*str*) – The name of the result.

Returns
The path to the result.

Return type
Path

Raises
ValueError – Raised if result does not exist.

get_models_directory

Project.**get_models_directory**() → Path

Get the path to the model directory of the project.

Returns
The path to the project's model directory.

Return type
Path

get_parameters_directory

Project.**get_parameters_directory**() → Path

Get the path to the parameter directory of the project.

Returns
The path to the project's parameter directory.

Return type
Path

get_result_path

Project.**get_result_path**(*result_name: str*, *, *latest: bool = False*) → Path

Get the path to a result with name `name`.

Parameters

- **result_name** (*str*) – The name of the result.
- **latest** (*bool*) – Flag to deactivate warning about using latest result. Defaults to False

Returns
The path to the result.

Return type
Path

Raises
ValueError – Raised if result does not exist.

import_data

Project.**import_data**(*dataset: LoadableDataset | Mapping[str, LoadableDataset]*, *dataset_name: str | None = None*, *allow_overwrite: bool = False*, *ignore_existing: bool = True*)

Import a dataset by saving it as an .nc file in the project's data folder.

Parameters

- **dataset** (*LoadableDataset*) – Dataset instance or path to a dataset.
- **dataset_name** (*str | None*) – The name of the dataset (needs to be provided when dataset is an xarray instance). Defaults to None.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing dataset.
- **ignore_existing** (*bool*) – Whether to skip import if the dataset already exists and allow_overwrite is False. Defaults to `True`.

load_data

Project.**load_data**(*dataset_name: str*, ***, *add_svd: bool = False*, *lsv_dim: Hashable = 'time'*, *rsv_dim: Hashable = 'spectral'*) → xr.Dataset

Load a dataset, with SVD data if `add_svd` is `True`.

Parameters

- **dataset_name** (*str*) – The name of the dataset.
- **add_svd** (*bool*) – Whether or not to calculate and add SVD data. Defaults to False.
- **lsv_dim** (*Hashable*) – Dimension of the left singular vectors. Defaults to "time".
- **rsv_dim** (*Hashable*) – Dimension of the right singular vectors. Defaults to "spectral",

Returns
The loaded dataset, with SVD data if `add_svd` is `True`.

Return type
xr.Dataset

Raises
ValueError – Raised if the dataset does not exist.

load_latest_result

Project.**load_latest_result**(*result_name: str*) → *Result*

Load a result.

Parameters
result_name (*str*) – The name of the result.

Returns
The loaded result.

Return type
Result

Raises
ValueError – Raised if result does not exist.

load_model

Project.**load_model**(*model_name: str*) → *Model*

Load a model.

Parameters
model_name (*str*) – The name of the model.

Returns
The loaded model.

Return type
Model

Raises
ValueError – Raised if the model does not exist.

load_parameters

Project.**load_parameters**(*parameters_name: str*) → *Parameters*

Load parameters.

Parameters
parameters_name (*str*) – The name of the parameters.

Raises
ValueError – Raised if parameters do not exist.

Returns
The loaded parameters.

Return type
Parameters

load_result

Project.**load_result**(*result_name: str*, *, *latest: bool = False*) → *Result*

Load a result.

Parameters

- **result_name** (*str*) – The name of the result.
- **latest** (*bool*) – Flag to deactivate warning about using latest result. Defaults to False

Returns
The loaded result.

Return type
Result

Raises
ValueError – Raised if result does not exist.

markdown

Project.**markdown**() → *MarkdownStr*

Format the project as a markdown text.

Returns
MarkdownStr – The markdown string.

Return type
str

open

classmethod Project.**open**(*project_folder_or_file: str | Path*, *create_if_not_exist: bool = True*) → *Project*

Open a new project.

Parameters

- **project_folder_or_file** (*str | Path*) – The path to a project folder or file.
- **create_if_not_exist** (*bool*) – Create the project if not existent.

Returns
The project instance.

Return type
Project

Raises
FileNotFoundError – Raised when the project file does not not exist and *create_if_not_exist* is *False*.

optimize

Project.**optimize**(*model_name: str*, *parameters_name: str*, *result_name: str | None = None*, *maximum_number_function_evaluations: int | None = None*, *clp_link_tolerance: float = 0.0*, *data_lookup_override: Mapping[str, LoadableDataset] | None = None*) → *Result*

Optimize a model.

Parameters

- **model_name** (*str*) – The model to optimize.
- **parameters_name** (*str*) – The initial parameters.
- **result_name** (*str | None*) – The name of the result.
- **maximum_number_function_evaluations** (*int | None*) – The maximum number of function evaluations.
- **clp_link_tolerance** (*float*) – The CLP link tolerance.
- **data_lookup_override** (*Mapping[str, LoadableDataset] | None*) – Allows to bypass the default dataset lookup in the project `data` folder and use a different dataset for the optimization without changing the model. This is especially useful when working with preprocessed data. Defaults to `None`.

Returns
Result of the optimization.

Return type
Result

show_model_definition

Project.**show_model_definition**(*model_name: str*, *syntax: str | None = None*) → *MarkdownStr*

Show model definition file content with syntax highlighting.

Parameters

- **model_name** (*str*) – The name of the model.
- **syntax** (*str | None*) – Syntax used for syntax highlighting. Defaults to None which means that the syntax is inferred based on the file extension. Pass the value "" to deactivate syntax highlighting.

Returns
Model definition file content with syntax highlighting to render in ipython.

Return type
MarkdownStr

show_parameters_definition

Project.**show_parameters_definition**(*parameters_name: str*, *syntax: str | None = None*, *, *as_dataframe: bool | None = None*) → *MarkdownStr* | DataFrame

Show parameters definition file content with syntax highlighting.

Parameters

- **parameters_name** (*str*) – The name of the parameters.
- **syntax** (*str | None*) – Syntax used for syntax highlighting. Defaults to None which means that the syntax is inferred based on the file extension. Pass the value "" to deactivate syntax highlighting.
- **as_dataframe** (*bool | None*) – Whether or not to show the `Parameters` definition as pandas.DataFrame (mostly useful for non string formats). Defaults to None which means that it will be inferred to `True` for known non string formats like `xlsx`.

Returns
Parameters definition file content with syntax highlighting to render in ipython.

Return type
MarkdownStr | pd.DataFrame

validate

Project.**validate**(*model_name: str*, *parameters_name: str | None = None*) → *MarkdownStr*

Check that the model is valid, list all issues in the model if there are any.

If `parameters_name` also consider the `Parameters` when validating.

Parameters

- **model_name** (*str*) – The name of the model to validate.

- **parameters_name** (*str | None*) – The name of the parameters to use when validating. Defaults to None which means that parameters are not considered when validating the model.

Returns
Text indicating if the model is valid or not.

Return type
MarkdownStr

Methods Documentation

static create(*folder: str | Path, allow_overwrite: bool = False*) → *Project*

Create a new project folder and file.

Parameters

- **folder** (*str | Path | None*) – The folder where the project will be created. If None, the current work directory will be used.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing project file.

Returns
The created project.

Return type
Project

Raises
FileExistsError – Raised if the project file already exists and *allow_overwrite=False*.

create_scheme(*model_name: str, parameters_name: str, maximum_number_function_evaluations: int | None = None, clp_link_tolerance: float = 0.0, data_lookup_override: Mapping[str, LoadableDataset] | None = None*) → *Scheme*

Create a scheme for optimization.

Parameters

- **model_name** (*str*) – The model to optimize.
- **parameters_name** (*str*) – The initial parameters.
- **maximum_number_function_evaluations** (*int | None*) – The maximum number of function evaluations.
- **clp_link_tolerance** (*float*) – The CLP link tolerance.
- **data_lookup_override** (*Mapping[str, LoadableDataset] | None*) – Allows to bypass the default dataset lookup in the project data folder and use a different dataset for the optimization without changing the model. This is especially useful when working with preprocessed data. Defaults to None.

Returns
The created scheme.

Return type
Scheme

property data: Mapping[str, Path]

Get all project datasets.

Returns
The models of the datasets.

Return type
Mapping[str, Path]

file: Path

folder: Path = None

generate_model(*model_name: str*, *generator_name: str*, *generator_arguments: dict[str, Any]*, ***, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate a model.

Parameters

- **model_name** (*str*) – The name of the model.
- **generator_name** (*str*) – The generator for the model.
- **generator_arguments** (*dict[str, Any]*) – Arguments for the generator.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing model.
- **ignore_existing** (*bool*) – Whether to ignore generation of a model file if it already exists.

generate_parameters(*model_name: str*, *parameters_name: str | None = None*, ***, *format_name: Literal['yml', 'yaml', 'csv'] = 'csv'*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate parameters for a model.

Parameters

- **model_name** (*str*) – The model.
- **parameters_name** (*str | None*) – The name of the parameters. If None it will be <model_name>_parameters.
- **format_name** (*Literal["yml", "yaml", "csv"]*) – The parameter format.
- **allow_overwrite** (*bool*) – Whether to overwrite existing parameters.
- **ignore_existing** (*bool*) – Whether to ignore generation of a parameter file if it already exists.

get_latest_result_path(*result_name: str*) → Path

Get the path to a result with name name.

Parameters
result_name (*str*) – The name of the result.

Returns
The path to the result.

Return type
Path

Raises
ValueError – Raised if result does not exist.

`get_models_directory`() → Path

Get the path to the model directory of the project.

Returns
The path to the project's model directory.

Return type
Path

`get_parameters_directory`() → Path

Get the path to the parameter directory of the project.

Returns
The path to the project's parameter directory.

Return type
Path

`get_result_path`(*result_name: str*, *, *latest: bool = False*) → Path

Get the path to a result with name `name`.

Parameters

- **result_name** (*str*) – The name of the result.
- **latest** (*bool*) – Flag to deactivate warning about using latest result. Defaults to False

Returns
The path to the result.

Return type
Path

Raises
ValueError – Raised if result does not exist.

`property has_data: bool`

Check if the project has datasets.

Returns
Whether the project has datasets.

Return type
bool

`property has_models: bool`

Check if the project has models.

Returns
Whether the project has models.

Return type
bool

`property has_parameters: bool`

Check if the project has parameters.

Returns
Whether the project has parameters.

Return type
bool

property has_results: bool

Check if the project has results.

Returns
Whether the project has results.

Return type
bool

import_data(*dataset: LoadableDataset | Mapping[str, LoadableDataset]*, *dataset_name: str | None = None*, *allow_overwrite: bool = False*, *ignore_existing: bool = True*)

Import a dataset by saving it as an .nc file in the project's data folder.

Parameters

- **dataset** (*LoadableDataset*) – Dataset instance or path to a dataset.
- **dataset_name** (*str | None*) – The name of the dataset (needs to be provided when dataset is an xarray instance). Defaults to None.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing dataset.
- **ignore_existing** (*bool*) – Whether to skip import if the dataset already exists and allow_overwrite is False. Defaults to `True`.

load_data(*dataset_name: str*, ***, *add_svd: bool = False*, *lsv_dim: Hashable = 'time'*, *rsv_dim: Hashable = 'spectral'*) → xr.Dataset

Load a dataset, with SVD data if `add_svd` is `True`.

Parameters

- **dataset_name** (*str*) – The name of the dataset.
- **add_svd** (*bool*) – Whether or not to calculate and add SVD data. Defaults to False.
- **lsv_dim** (*Hashable*) – Dimension of the left singular vectors. Defaults to "time".
- **rsv_dim** (*Hashable*) – Dimension of the right singular vectors. Defaults to "spectral",

Returns
The loaded dataset, with SVD data if `add_svd` is `True`.

Return type
xr.Dataset

Raises
ValueError – Raised if the dataset does not exist.

load_latest_result(*result_name: str*) → *Result*

Load a result.

Parameters
result_name (*str*) – The name of the result.

Returns
The loaded result.

Return type
Result

Raises
ValueError – Raised if result does not exist.

load_model(*model_name: str*) → *Model*

Load a model.

Parameters
model_name (str) – The name of the model.

Returns
The loaded model.

Return type
Model

Raises
ValueError – Raised if the model does not exist.

load_parameters(*parameters_name: str*) → *Parameters*

Load parameters.

Parameters
parameters_name (str) – The name of the parameters.

Raises
ValueError – Raised if parameters do not exist.

Returns
The loaded parameters.

Return type
Parameters

load_result(*result_name: str*, *, *latest: bool = False*) → *Result*

Load a result.

Parameters

- result_name (str) – The name of the result.
- latest (bool) – Flag to deactivate warning about using latest result. Defaults to False

Returns
The loaded result.

Return type
Result

Raises
ValueError – Raised if result does not exist.

markdown() → *MarkdownStr*

Format the project as a markdown text.

Returns
MarkdownStr – The markdown string.

Return type
str

property models: Mapping[str, Path]

Get all project models.

Returns
The models of the project.

Return type
Mapping[str, Path]

classmethod open(*project_folder_or_file: str | Path*, *create_if_not_exist: bool = True*) → *Project*

Open a new project.

Parameters

- **project_folder_or_file** (*str | Path*) – The path to a project folder or file.
- **create_if_not_exist** (*bool*) – Create the project if not existent.

Returns
The project instance.

Return type
Project

Raises
FileNotFoundError – Raised when the project file does not not exist and *create_if_not_exist* is *False*.

optimize(*model_name: str*, *parameters_name: str*, *result_name: str | None = None*, *maximum_number_function_evaluations: int | None = None*, *clp_link_tolerance: float = 0.0*, *data_lookup_override: Mapping[str, LoadableDataset] | None = None*) → *Result*

Optimize a model.

Parameters

- **model_name** (*str*) – The model to optimize.
- **parameters_name** (*str*) – The initial parameters.
- **result_name** (*str | None*) – The name of the result.
- **maximum_number_function_evaluations** (*int | None*) – The maximum number of function evaluations.
- **clp_link_tolerance** (*float*) – The CLP link tolerance.
- **data_lookup_override** (*Mapping[str, LoadableDataset] | None*) – Allows to bypass the default dataset lookup in the project `data` folder and use a different dataset for the optimization without changing the model. This is especially useful when working with preprocessed data. Defaults to `None`.

Returns
Result of the optimization.

Return type
Result

property parameters: Mapping[str, Path]

Get all project parameters.

Returns
The parameters of the project.

Return type
Mapping[str, Path]

property results: Mapping[str, Path]

Get all project results.

Returns
The results of the project.

Return type
Mapping[str, Path]

show_model_definition(*model_name: str*, *syntax: str | None = None*) → *MarkdownStr*

Show model definition file content with syntax highlighting.

Parameters

- **model_name** (*str*) – The name of the model.
- **syntax** (*str | None*) – Syntax used for syntax highlighting. Defaults to None which means that the syntax is inferred based on the file extension. Pass the value "" to deactivate syntax highlighting.

Returns
Model definition file content with syntax highlighting to render in ipython.

Return type
MarkdownStr

show_parameters_definition(*parameters_name: str*, *syntax: str | None = None*, ***, *as_dataframe: bool | None = None*) → *MarkdownStr* | DataFrame

Show parameters definition file content with syntax highlighting.

Parameters

- **parameters_name** (*str*) – The name of the parameters.
- **syntax** (*str | None*) – Syntax used for syntax highlighting. Defaults to None which means that the syntax is inferred based on the file extension. Pass the value "" to deactivate syntax highlighting.
- **as_dataframe** (*bool | None*) – Whether or not to show the `Parameters` definition as pandas.DataFrame (mostly useful for non string formats). Defaults to None which means that it will be inferred to `True` for known non string formats like `xlsx`.

Returns
Parameters definition file content with syntax highlighting to render in ipython.

Return type
MarkdownStr | pd.DataFrame

validate(*model_name: str*, *parameters_name: str | None = None*) → *MarkdownStr*

Check that the model is valid, list all issues in the model if there are any.

If `parameters_name` also consider the `Parameters` when validating.

Parameters

- **model_name** (*str*) – The name of the model to validate.
- **parameters_name** (*str | None*) – The name of the parameters to use when validating. Defaults to None which means that parameters are not considered when validating the model.

Returns
Text indicating if the model is valid or not.

Return type
MarkdownStr

version: str

project_data_registry

The glotaran data registry module.

Classes

Summary

ProjectDataRegistry	A registry for data.

ProjectDataRegistry

class glotaran.project.project_data_registry.**ProjectDataRegistry**(*directory: Path*)

Bases: *ProjectRegistry*

A registry for data.

Initialize a data registry.

Parameters
directory (*Path*) – The registry directory.

Attributes Summary

directory	Get the registry directory.
empty	Whether the registry is empty.
items	Get the items of the registry.

directory

ProjectDataRegistry.**directory**

Get the registry directory.

Returns
The registry directory.

Return type
Path

empty

ProjectDataRegistry.**empty**

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

items

ProjectDataRegistry.**items**

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

Methods Summary

import_data	Import a dataset.
is_item	Check if the path contains an registry item.
load_item	Load an registry item by it's name.
markdown	Format the registry items as a markdown text.

import_data

ProjectDataRegistry.**import_data**(*dataset: str | Path | Dataset | DataArray*, *dataset_name: str | None = None*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Import a dataset.

Parameters

- **dataset** (*str | Path | xr.Dataset | xr.DataArray*) – Dataset instance or path to a dataset.

- **dataset_name** (*str | None*) – The name of the dataset (needs to be provided when dataset is an xarray instance). Defaults to None.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing dataset.
- **ignore_existing** (*bool*) – Whether to ignore import if the dataset already exists.

Raises
ValueError – When importing from xarray object and not providing a name.

is_item

ProjectDataRegistry.**is_item**(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

load_item

ProjectDataRegistry.**load_item**(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown

ProjectDataRegistry.**markdown**(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

Methods Documentation

property directory: Path

Get the registry directory.

Returns
The registry directory.

Return type
Path

property empty: bool

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

import_data(*dataset: str | Path | Dataset | DataArray*, *dataset_name: str | None = None*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Import a dataset.

Parameters

- **dataset** (*str | Path | xr.Dataset | xr.DataArray*) – Dataset instance or path to a dataset.
- **dataset_name** (*str | None*) – The name of the dataset (needs to be provided when dataset is an xarray instance). Defaults to None.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing dataset.
- **ignore_existing** (*bool*) – Whether to ignore import if the dataset already exists.

Raises
ValueError – When importing from xarray object and not providing a name.

is_item(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

property items: *ItemMapping*

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

load_item(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

project_model_registry

The glotaran model registry module.

Classes

Summary

ProjectModelRegistry	A registry for models.

ProjectModelRegistry

class glotaran.project.project_model_registry.**ProjectModelRegistry**(*directory: Path*)

Bases: *ProjectRegistry*

A registry for models.

Initialize a model registry.

Parameters
directory (*Path*) – The registry directory.

Attributes Summary

directory	Get the registry directory.
empty	Whether the registry is empty.
items	Get the items of the registry.

directory

ProjectModelRegistry.**directory**

Get the registry directory.

Returns
The registry directory.

Return type
Path

empty

ProjectModelRegistry.**empty**

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

items

ProjectModelRegistry.**items**

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

Methods Summary

generate_model	Generate a model.
is_item	Check if the path contains an registry item.
load_item	Load an registry item by it's name.
markdown	Format the registry items as a markdown text.

generate_model

ProjectModelRegistry.**generate_model**(*name: str*, *generator_name: str*, *generator_arguments:* GeneratorArguments, *, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate a model.

Parameters

- **name** (*str*) – The name of the model.
- **generator_name** (*str*) – The generator for the model.
- **generator_arguments** (GeneratorArguments) – Arguments for the generator.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing model.
- **ignore_existing** (*bool*) – Whether to ignore generation of a model file if it already exists.

Raises

FileExistsError – Raised if model is already existing and *allow_overwrite=False*.

is_item

ProjectModelRegistry.**is_item**(*path: Path*) → bool

Check if the path contains an registry item.

Parameters

path (*Path*) – The path to check.

Returns

Whether the path contains an item.

Return type

bool

load_item

ProjectModelRegistry.**load_item**(*name: str*) → Any

Load an registry item by it's name.

Parameters

name (*str*) – The item name.

Returns

The loaded item.

Return type

Any

Raises

ValueError – Raise if the item does not exist.

markdown

ProjectModelRegistry.**markdown**(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

Methods Documentation

property directory: Path

Get the registry directory.

Returns
The registry directory.

Return type
Path

property empty: bool

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

generate_model(*name: str*, *generator_name: str*, *generator_arguments:* GeneratorArguments, *, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate a model.

Parameters

- **name** (*str*) – The name of the model.
- **generator_name** (*str*) – The generator for the model.
- **generator_arguments** (GeneratorArguments) – Arguments for the generator.
- **allow_overwrite** (*bool*) – Whether to overwrite an existing model.
- **ignore_existing** (*bool*) – Whether to ignore generation of a model file if it already exists.

Raises
FileExistsError – Raised if model is already existing and *allow_overwrite=False*.

is_item(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

property items: *ItemMapping*

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

load_item(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

project_parameter_registry

The glotaran parameter registry module.

Classes

Summary

ProjectParameterRegistry	A registry for parameters.

ProjectParameterRegistry

class glotaran.project.project_parameter_registry.**ProjectParameterRegistry**(*directory: Path*)

Bases: *ProjectRegistry*

A registry for parameters.

Initialize a parameter registry.

Parameters
directory (*Path*) – The registry directory.

Attributes Summary

directory	Get the registry directory.
empty	Whether the registry is empty.
items	Get the items of the registry.

directory

ProjectParameterRegistry.**directory**

Get the registry directory.

Returns
The registry directory.

Return type
Path

empty

ProjectParameterRegistry.**empty**

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

items

ProjectParameterRegistry.**items**

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

Methods Summary

generate_parameters	Generate parameters for a model.
is_item	Check if the path contains an registry item.
load_item	Load an registry item by it's name.
markdown	Format the registry items as a markdown text.

generate_parameters

ProjectParameterRegistry.**generate_parameters**(*model:* Model, *name: str | None*, ***, *format_name: Literal['yml', 'yaml', 'csv'] = 'csv'*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate parameters for a model.

Parameters

- **model** (Model) – The model.
- **name** (*str | None*) – The name of the parameters.
- **format_name** (*Literal["yml", "yaml", "csv"]*) – The parameter format.
- **allow_overwrite** (*bool*) – Whether to overwrite existing parameters.
- **ignore_existing** (*bool*) – Whether to ignore generation of a parameter file if it already exists.

Raises
FileExistsError – Raised if parameters is already existing and *allow_overwrite=False*.

is_item

ProjectParameterRegistry.**is_item**(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

load_item

ProjectParameterRegistry.**load_item**(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown

ProjectParameterRegistry.**markdown**(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

Methods Documentation

property directory: Path

Get the registry directory.

Returns
The registry directory.

Return type
Path

property empty: bool

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

generate_parameters(*model: Model*, *name: str | None*, ***, *format_name: Literal['yml', 'yaml', 'csv'] = 'csv'*, *allow_overwrite: bool = False*, *ignore_existing: bool = False*)

Generate parameters for a model.

Parameters

- **model** (*Model*) – The model.
- **name** (*str | None*) – The name of the parameters.
- **format_name** (*Literal["yml", "yaml", "csv"]*) – The parameter format.
- **allow_overwrite** (*bool*) – Whether to overwrite existing parameters.
- **ignore_existing** (*bool*) – Whether to ignore generation of a parameter file if it already exists.

Raises

FileExistsError – Raised if parameters is already existing and *allow_overwrite=False*.

is_item(*path: Path*) → bool

Check if the path contains an registry item.

Parameters

path (*Path*) – The path to check.

Returns

Whether the path contains an item.

Return type

bool

property items: *ItemMapping*

Get the items of the registry.

Returns

The items of the registry.

Return type

ItemMapping

load_item(*name: str*) → Any

Load an registry item by it's name.

Parameters

name (*str*) – The item name.

Returns

The loaded item.

Return type

Any

Raises

ValueError – Raise if the item does not exist.

markdown(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

project_registry

The glotaran registry module.

Classes

Summary

ItemMapping	Container class for `ProjectRegistry` items.
ProjectRegistry	A registry base class.

ItemMapping

class glotaran.project.project_registry.**ItemMapping**(*data: Mapping[str, Path]*, *item_name: str*)

Bases: `Mapping`

Container class for `ProjectRegistry` items.

The main purpose of this class is to show a user friendly error when accessing none existing items.

Initialize class instance as wrapper around `data`.

Parameters

- **data** (*Mapping[str, Path]*) – Underlying data that are used for mapping.
- **item_name** (*str*) – Name of items in the registry used to format warning and exception (e.g. 'Parameters').

Methods Summary

get
items
keys
values

get

ItemMapping.**get**(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items

ItemMapping.**items**() → a set-like object providing a view on D's items

keys

ItemMapping.**keys**() → a set-like object providing a view on D's keys

values

ItemMapping.**values**() → an object providing a view on D's values

Methods Documentation

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

ProjectRegistry

class glotaran.project.project_registry.**ProjectRegistry**(*directory: Path*, *file_suffix: str | Iterable[str]*, *loader: Callable*, *item_name: str*)

Bases: object

A registry base class.

Initialize a registry.

Parameters

- **directory** (*Path*) – The registry directory.
- **file_suffix** (*str | Iterable[str]*) – The suffixes of item files.
- **loader** (*Callable*) – A loader for the registry items.
- **item_name** (*str*) – Name of items in the registry used to format warning and exception (e.g. 'Parameters').

Attributes Summary

directory	Get the registry directory.
empty	Whether the registry is empty.
items	Get the items of the registry.

directory

ProjectRegistry.**directory**

Get the registry directory.

Returns
The registry directory.

Return type
Path

empty

ProjectRegistry.**empty**

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

items

ProjectRegistry.**items**

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

Methods Summary

is_item	Check if the path contains an registry item.
load_item	Load an registry item by it's name.
markdown	Format the registry items as a markdown text.

is_item

ProjectRegistry.**is_item**(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

load_item

ProjectRegistry.**load_item**(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown

ProjectRegistry.**markdown**(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

Methods Documentation

property directory: Path

Get the registry directory.

Returns
The registry directory.

Return type
Path

property empty: bool

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

is_item(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

property items: *ItemMapping*

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

load_item(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the

parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

Exceptions

Exception Summary

`AmbiguousNameWarning`	Warning thrown when an item with the same name already exists.

AmbiguousNameWarning

exception glotaran.project.project_registry.**AmbiguousNameWarning**(*, *item_name: str*, *items: dict[str, pathlib.Path]*, *item_key: str*, *unique_item_key: str*, *project_root_path: Path*)

Warning thrown when an item with the same name already exists.

This is the case if two files with the same name but different extensions exist next to each other.

Initialize `AmbiguousNameWarning` with a formatted message.

Parameters

- **item_name** (*str*) – Name of items in the registry (e.g. 'Parameters').
- **items** (*dict[str, Path]*) – Known items at this iteration point.
- **item_key** (*str*) – Key that would have been used if the file names weren't ambiguous.
- **unique_item_key** (*str*) – Unique key for the item with ambiguous file name.
- **project_root_path** (*Path*) – Root path of the project.

project_result_registry

The glotaran result registry module.

Classes

Summary

ProjectResultRegistry	A registry for results.

ProjectResultRegistry

class glotaran.project.project_result_registry.**ProjectResultRegistry**(*directory: Path*)

Bases: *ProjectRegistry*

A registry for results.

Initialize a result registry.

Parameters
directory (*Path*) – The registry directory.

Attributes Summary

directory	Get the registry directory.
empty	Whether the registry is empty.
items	Get the items of the registry.
result_pattern	

directory

ProjectResultRegistry.**directory**

Get the registry directory.

Returns
The registry directory.

Return type
Path

empty

ProjectResultRegistry.**empty**

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

items

ProjectResultRegistry.**items**

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

result_pattern

ProjectResultRegistry.**result_pattern = re.compile('.+_run_\\d{4}$')**

Methods Summary

create_result_run_name	Create a result name for a model.
is_item	Check if the path contains an registry item.
load_item	Load an registry item by it's name.
markdown	Format the registry items as a markdown text.
previous_result_paths	List previous result paths with base_name.
save	Save a result.

create_result_run_name

ProjectResultRegistry.**create_result_run_name**(*base_name: str*) → str

Create a result name for a model.

Parameters
base_name (*str*) – The base name for the result provided by user or derived from model name.

Returns
Folder name for the new result to be saved in.

Return type
str

is_item

ProjectResultRegistry.**is_item**(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

load_item

ProjectResultRegistry.**load_item**(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

markdown

ProjectResultRegistry.**markdown**(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

previous_result_paths

ProjectResultRegistry.**previous_result_paths**(*base_name: str*) → list[pathlib.Path]

List previous result paths with base_name.

Parameters
base_name (*str*) – The base name for the result provided by user or derived from model name.

Returns
Paths to previous results with name base_name.

Return type
list[Path]

save

ProjectResultRegistry.**save**(*name: str*, *result:* Result)

Save a result.

Parameters

- **name** (*str*) – The name of the result.
- **result** (Result) – The result to save.

Methods Documentation

create_result_run_name(*base_name: str*) → str

Create a result name for a model.

Parameters
base_name (*str*) – The base name for the result provided by user or derived from model name.

Returns
Folder name for the new result to be saved in.

Return type
str

property directory: Path

Get the registry directory.

Returns
The registry directory.

Return type
Path

property empty: bool

Whether the registry is empty.

Returns
Whether the registry is empty.

Return type
bool

`is_item`(*path: Path*) → bool

Check if the path contains an registry item.

Parameters
path (*Path*) – The path to check.

Returns
Whether the path contains an item.

Return type
bool

`property items: ItemMapping`

Get the items of the registry.

Returns
The items of the registry.

Return type
ItemMapping

`load_item`(*name: str*) → Any

Load an registry item by it's name.

Parameters
name (*str*) – The item name.

Returns
The loaded item.

Return type
Any

Raises
ValueError – Raise if the item does not exist.

`markdown`(*join_indentation: int = 0*) → *MarkdownStr*

Format the registry items as a markdown text.

Parameters
join_indentation (*int*) – Number of whitespaces to indent when joining the parts. This is intended to be used with dedent when used in an indented f-string. Defaults to 0.

Returns
MarkdownStr – The markdown string.

Return type
str

`previous_result_paths`(*base_name: str*) → list[pathlib.Path]

List previous result paths with base_name.

Parameters
base_name (*str*) – The base name for the result provided by user or derived from model name.

Returns
Paths to previous results with name `base_name`.

Return type
list[Path]

```
result_pattern = re.compile('.+_run_\\d{4}$')
```

save(*name: str*, *result:* Result)

Save a result.

Parameters

- **name** (*str*) – The name of the result.
- **result** (Result) – The result to save.

result

The result class for global analysis.

Classes

Summary

Result	The result of a global analysis.

Result

class glotaran.project.result.**Result**(*number_of_function_evaluations: int*, *success: bool*, *termination_reason: str*, *glotaran_version: str*, *free_parameter_labels: list[str]*, *scheme:* Scheme, *initial_parameters:* Parameters, *optimized_parameters:* Parameters, *parameter_history:* ParameterHistory, *optimization_history:* OptimizationHistory, *data: Mapping[str, xr.Dataset]*, *additional_penalty: list[np.ndarray] | None = None*, *cost: ArrayLike | None = None*, *chi_square: float | None = None*, *covariance_matrix: ArrayLike | None = None*, *degrees_of_freedom: int | None = None*, *number_of_clps: int | None = None*, *jacobian: ArrayLike | list | None = None*, *number_of_residuals: int | None = None*, *number_of_jacobian_evaluations: int | None = None*, *number_of_free_parameters: int | None = None*, *optimality: float | None = None*, *reduced_chi_square: float | None = None*, *root_mean_square_error: float | None = None*)

Bases: object

The result of a global analysis.

Attributes Summary

additional_penalty	A vector with the value for each additional penalty, or None
chi_square	The chi-square of the optimization.
cost	The final cost.
covariance_matrix	Covariance matrix.
degrees_of_freedom	Degrees of freedom in optimization $N - N_{vars} - N_{clps}$.
jacobian	Modified Jacobian matrix at the solution
model	Return the model used to fit result.
number_of_clps	Number of conditionally linear parameters N_{clps}.
number_of_data_points	Return the number of values in the residual vector N.
number_of_free_parameters	Number of free parameters in optimization N_{vars}
number_of_jacobian_evaluations	The number of jacobian evaluations.
number_of_parameters	Return the number of free parameters in optimization N_{vars}.
number_of_residuals	Number of values in the residual vector N.
optimality	
reduced_chi_square	The reduced chi-square of the optimization.
root_mean_square_error	The root mean square error the optimization.
source_path	
number_of_function_evaluations	The number of function evaluations.
success	Indicates if the optimization was successful.
termination_reason	The reason (message when) the optimizer terminated
glotaran_version	The glotaran version used to create the result.
free_parameter_labels	List of labels of the free parameters used in optimization.
scheme	
initial_parameters	
optimized_parameters	
parameter_history	The parameter history.
optimization_history	The optimization history.
data	The resulting data as a dictionary of xarray.Dataset.

additional_penalty

Result.**additional_penalty: list[np.ndarray] | None = None**

A vector with the value for each additional penalty, or None

chi_square

Result.**chi_square: float | None = None**

The chi-square of the optimization.

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

cost

Result.**cost: ArrayLike | None = None**

The final cost.

covariance_matrix

Result.**covariance_matrix: ArrayLike | None = None**

Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

degrees_of_freedom

Result.**degrees_of_freedom: int | None = None**

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

jacobian

Result.**jacobian: ArrayLike | list | None = None**

Modified Jacobian matrix at the solution

See also: `scipy.optimize.least_squares()`

model

Result.**model**

Return the model used to fit result.

Returns
The model instance.

Return type
Model

number_of_clps

Result.**number_of_clps**: int | None = None

Number of conditionally linear parameters N_{clps}.

number_of_data_points

Result.**number_of_data_points**

Return the number of values in the residual vector N.

Deprecated since it returned the wrong value.

Returns
Number of values in the residual vector N.

Return type
int | None

number_of_free_parameters

Result.**number_of_free_parameters**: int | None = None

Number of free parameters in optimization N_{vars}

number_of_jacobian_evaluations

Result.**number_of_jacobian_evaluations**: int | None = None

The number of jacobian evaluations.

number_of_parameters

Result.**number_of_parameters**

Return the number of free parameters in optimization N_{vars}.

Returns
Number of free parameters in optimization N_{vars}.

Return type
int | None

number_of_residuals

Result.**number_of_residuals**: int | None = None

Number of values in the residual vector N.

optimality

Result.**optimality: float | None = None**

reduced_chi_square

Result.**reduced_chi_square: float | None = None**

The reduced chi-square of the optimization.

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

root_mean_square_error

Result.**root_mean_square_error: float | None = None**

The root mean square error the optimization.

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

source_path

Result.**source_path: StrOrPath = 'result.yml'**

number_of_function_evaluations

Result.**number_of_function_evaluations: int**

The number of function evaluations.

success

Result.**success: bool**

Indicates if the optimization was successful.

termination_reason

Result.**termination_reason: str**

The reason (message when) the optimizer terminated

glotaran_version

Result.**glotaran_version: str**

The glotaran version used to create the result.

free_parameter_labels

Result.**free_parameter_labels**: list[str]

List of labels of the free parameters used in optimization.

scheme

Result.**scheme**: *Scheme*

initial_parameters

Result.**initial_parameters**: *Parameters*

optimized_parameters

Result.**optimized_parameters**: *Parameters*

parameter_history

Result.**parameter_history**: *ParameterHistory*

The parameter history.

optimization_history

Result.**optimization_history**: *OptimizationHistory*

The optimization history.

data

Result.**data**: Mapping[str, xr.Dataset]

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.

Methods Summary

create_clp_guide_dataset	Create dataset for clp guidance.
get_scheme	Return a new scheme from the Result object with optimized parameters.
loader	Create a *Result* instance from the specs defined in a file.
markdown	Format the model as a markdown text.
recreate	Recrate a result from the initial parameters.
save	Save the result to given folder.
verify	Verify a result.

create_clp_guide_dataset

Result.**create_clp_guide_dataset**(*clp_label: str*, *dataset_name: str*) → Dataset

Create dataset for clp guidance.

Parameters

- **clp_label** (*str*) – Label of the clp to guide.
- **dataset_name** (*str*) – Name of dataset to extract the guide from.

Returns

DataArray containing the clp guide, with `clp_label` dimension replaced by the model dimensions first value.

Return type

xr.Dataset

Raises

- **ValueError** – If `dataset_name` is not in result.
- **ValueError** – If `clp_labels` is not in result.

Examples

Extracting the clp guide from an optimization result object.

```
from glotaran.io import save_dataset

clp_guide = result.create_clp_guide_dataset("species_1", "dataset_1")
save_dataset(clp_guide, "clp_guide__result_dataset_1__species_1.nc")
```

get_scheme

`Result.get_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

loader

`Result.loader(format_name: str | None = None, **kwargs: Any) → Result`

Create a `Result` instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str | None*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns
`Result` instance created from the saved format.

Return type
Result

markdown

`Result.markdown(with_model: bool = True, *, base_heading_level: int = 1, wrap_model_in_details: bool = False) → MarkdownStr`

Format the model as a markdown text.

Parameters

- **with_model** (*bool*) – If *True*, the model will be printed with initial and optimized parameters filled in.
- **base_heading_level** (*int*) – The level of the base heading.
- **wrap_model_in_details** (*bool*) – Wraps model into details tag. Defaults to `False`

Returns
MarkdownStr – The scheme as markdown string.

Return type
str

recreate

Result.**recreate**() → *Result*

Recrate a result from the initial parameters.

Returns
The recreated result.

Return type
Result

save

Result.**save**(*path: StrOrPath*, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to given folder.

Parameters

- **path** (*StrOrPath*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for the saved result.

Returns
Paths to all the saved files.

Return type
list[str]

verify

Result.**verify**() → bool

Verify a result.

Returns
Weather the recreated result is equal to this result.

Return type
bool

Methods Documentation

additional_penalty: list[np.ndarray] | None = None

A vector with the value for each additional penalty, or None

chi_square: float | None = None

The chi-square of the optimization.

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

cost: ArrayLike | None = None

The final cost.

covariance_matrix: ArrayLike | None = None

Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

create_clp_guide_dataset(*clp_label: str*, *dataset_name: str*) → Dataset

Create dataset for clp guidance.

Parameters

- **clp_label** (*str*) – Label of the clp to guide.
- **dataset_name** (*str*) – Name of dataset to extract the guide from.

Returns

DataArray containing the clp guide, with `clp_label` dimension replaced by the model dimensions first value.

Return type

xr.Dataset

Raises

- **ValueError** – If `dataset_name` is not in result.
- **ValueError** – If `clp_labels` is not in result.

Examples

Extracting the clp guide from an optimization result object.

```
from glotaran.io import save_dataset

clp_guide = result.create_clp_guide_dataset("species_1", "dataset_1")
save_dataset(clp_guide, "clp_guide__result_dataset_1__species_1.nc")
```

data: Mapping[str, xr.Dataset]

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: int | None = None

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

free_parameter_labels: list[str]

List of labels of the free parameters used in optimization.

get_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

glotaran_version: str

The glotaran version used to create the result.

initial_parameters: *Parameters*

jacobian: ArrayLike | list | None = None

Modified Jacobian matrix at the solution

See also: `scipy.optimize.least_squares()`

loader(*format_name: str | None = None*, ***kwargs: Any*) → *Result*

Create a *Result* instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str | None*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns

Result instance created from the saved format.

Return type

Result

markdown(*with_model: bool = True*, ***, *base_heading_level: int = 1*, *wrap_model_in_details: bool = False*) → *MarkdownStr*

Format the model as a markdown text.

Parameters

- **with_model** (*bool*) – If *True*, the model will be printed with initial and optimized parameters filled in.
- **base_heading_level** (*int*) – The level of the base heading.
- **wrap_model_in_details** (*bool*) – Wraps model into details tag. Defaults to `False`

Returns

MarkdownStr – The scheme as markdown string.

Return type

str

property model: *Model*

Return the model used to fit result.

Returns

The model instance.

Return type

Model

number_of_clps: int | None = None

Number of conditionally linear parameters N_{clps}.

property number_of_data_points: int | None

Return the number of values in the residual vector N.

Deprecated since it returned the wrong value.

Returns
Number of values in the residual vector N.

Return type
int | None

number_of_free_parameters: int | None = None

Number of free parameters in optimization N_{vars}

number_of_function_evaluations: int

The number of function evaluations.

number_of_jacobian_evaluations: int | None = None

The number of jacobian evaluations.

property number_of_parameters: int | None

Return the number of free parameters in optimization N_{vars}.

Returns
Number of free parameters in optimization N_{vars}.

Return type
int | None

number_of_residuals: int | None = None

Number of values in the residual vector N.

optimality: float | None = None

optimization_history: *OptimizationHistory*

The optimization history.

optimized_parameters: *Parameters*

parameter_history: *ParameterHistory*

The parameter history.

recreate() → *Result*

Recrate a result from the initial parameters.

Returns
The recreated result.

Return type
Result

reduced_chi_square: float | None = None

The reduced chi-square of the optimization.

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

root_mean_square_error: float | None = None

The root mean square error the optimization.

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

save(*path: StrOrPath*, *saving_options:* SavingOptions = *SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to given folder.

Parameters

- **path** (*StrOrPath*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for the saved result.

Returns
Paths to all the saved files.

Return type
list[str]

scheme: *Scheme*

source_path: StrOrPath = 'result.yml'

success: bool

Indicates if the optimization was successful.

termination_reason: str

The reason (message when) the optimizer terminated

verify() → bool

Verify a result.

Returns
Weather the recreated result is equal to this result.

Return type
bool

scheme

The module for :class:Scheme.

Classes

Summary

Scheme	A scheme is a collection of a model, parameters and a dataset.

Scheme

class glotaran.project.scheme.**Scheme**(*model: Model*, *parameters: Parameters*, *data: Mapping[str, xr.Dataset]*, *clp_link_tolerance: float = 0.0*, *clp_link_method: Literal['nearest', 'backward', 'forward'] = 'nearest'*, *maximum_number_function_evaluations: int | None = None*, *add_svd: bool = True*, *ftol: float = 1e-08*, *gtol: float = 1e-08*, *xtol: float = 1e-08*, *optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'*, *result_path: str | None = None*)

Bases: object

A scheme is a collection of a model, parameters and a dataset.

A scheme also holds options for optimization.

Attributes Summary

add_svd
clp_link_method
clp_link_tolerance
ftol
gtol
maximum_number_function_evaluations
optimization_method
result_path
source_path
xtol
model
parameters
data

add_svd

Scheme.**add_svd: bool = True**

clp_link_method

Scheme.**clp_link_method: Literal['nearest', 'backward', 'forward'] = 'nearest'**

clp_link_tolerance

Scheme.**clp_link_tolerance: float = 0.0**

ftol

Scheme.**ftol: float = 1e-08**

gtol

Scheme.**gtol: float = 1e-08**

maximum_number_function_evaluations

Scheme.**maximum_number_function_evaluations: int | None = None**

optimization_method

Scheme.**optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'**

result_path

Scheme.**result_path: str | None = None**

source_path

Scheme.**source_path: StrOrPath = 'scheme.yml'**

xtol

Scheme.**xtol**: float = 1e-08

model

Scheme.**model**: *Model*

parameters

Scheme.**parameters**: *Parameters*

data

Scheme.**data**: Mapping[str, xr.Dataset]

Methods Summary

loader	Create a *Scheme* instance from the specs defined in a file.
markdown	Format the *Scheme* as markdown string.
valid	Check if there are no problems with the model or the parameters.
validate	Return a string listing all problems in the model and missing parameters.

loader

Scheme.**loader**(*format_name: str | None = None*, ***kwargs: Any*) → *Scheme*

Create a *Scheme* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns

Scheme instance created from the file.

Return type

Scheme

markdown

Scheme.**markdown**()

Format the *Scheme* as markdown string.

Returns
The scheme as markdown string.

Return type
MarkdownStr

valid

Scheme.**valid**() → bool

Check if there are no problems with the model or the parameters.

Returns
Whether the scheme is valid.

Return type
bool

validate

Scheme.**validate**() → *MarkdownStr*

Return a string listing all problems in the model and missing parameters.

Returns
A user-friendly string containing all the problems of a model if any. Defaults to 'Your model is valid.' if no problems are found.

Return type
MarkdownStr

Methods Documentation

add_svd: bool = True

clp_link_method: Literal['nearest', 'backward', 'forward'] = 'nearest'

clp_link_tolerance: float = 0.0

data: Mapping[str, xr.Dataset]

ftol: float = 1e-08

gtol: float = 1e-08

loader(*format_name: str | None = None*, ***kwargs: Any*) → *Scheme*

Create a *Scheme* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.

- **format_name** (*str | None*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns
Scheme instance created from the file.

Return type
Scheme

markdown()

Format the *Scheme* as markdown string.

Returns
The scheme as markdown string.

Return type
MarkdownStr

maximum_number_function_evaluations: int | None = None

model: *Model*

optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'

parameters: *Parameters*

result_path: str | None = None

source_path: StrOrPath = 'scheme.yml'

valid() → bool

Check if there are no problems with the model or the parameters.

Returns
Whether the scheme is valid.

Return type
bool

validate() → *MarkdownStr*

Return a string listing all problems in the model and missing parameters.

Returns
A user-friendly string containing all the problems of a model if any. Defaults to 'Your model is valid.' if no problems are found.

Return type
MarkdownStr

xtol: float = 1e-08

16.1.11 simulation

Package containing code for simulation of dataset models.

Modules

glotaran.simulation.simulation	Functions for simulating a dataset using a global optimization model.

simulation

Functions for simulating a dataset using a global optimization model.

Functions

Summary

simulate	Simulate a dataset using a model.
simulate_from_clp	Simulate a dataset model from pre-defined conditionally linear parameters.
simulate_full_model	Simulate a dataset model with global megacomplexes.

simulate

glotaran.simulation.simulation.**simulate**(*model:* Model, *dataset: str*, *parameters:* Parameters, *coordinates: dict[str, ArrayLike]*, *clp: xr.DataArray | None = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int | None = None*) → xr.Dataset

Simulate a dataset using a model.

Parameters

- **model** (Model) – The model containing the dataset model.
- **dataset** (*str*) – Label of the dataset to simulate
- **parameters** (Parameters) – The parameters for the simulation.
- **coordinates** (*dict[str, ArrayLike]*) – A dictionary with the coordinates used for simulation (e.g. time, wavelengths, ...).
- **clp** (*xr.DataArray | None*) – A matrix with conditionally linear parameters (e.g. spectra, pixel intensity, ...). Will be used instead of the dataset's global megacomplexes if not None.
- **noise** (*bool*) – Add noise to the simulation.
- **noise_std_dev** (*float*) – The standard deviation for noise simulation.
- **noise_seed** (*int | None*) – The seed for the noise simulation.

Returns
The simulated dataset.

Return type
xr.Dataset

Raises
`ValueError` – Raised if dataset model has no global megacomplex and no clp are provided.

simulate_from_clp

glotaran.simulation.simulation.**simulate_from_clp**(*dataset_model: DatasetModel*, *global_dimension: str*, *global_axis: ArrayLike*, *model_dimension: str*, *model_axis: ArrayLike*, *clp: xr.DataArray*) → xr.Dataset

Simulate a dataset model from pre-defined conditionally linear parameters.

Parameters

- **dataset_model** (`DatasetModel`) – The dataset model to simulate.
- **global_dimension** (*str*) – The global dimension of the dataset.
- **global_axis** (*ArrayLike*) – The global axis of the dataset.
- **model_dimension** (*str*) – The model dimension of the dataset.
- **model_axis** (*ArrayLike*) – The model axis of the dataset.
- **clp** (*xr.DataArray*) – A matrix with conditionally linear parameters.

Returns
The simulated dataset.

Return type
xr.Dataset

Raises
`ValueError` – Raised if the clp are missing the dimension 'clp_label'.

simulate_full_model

glotaran.simulation.simulation.**simulate_full_model**(*dataset_model: DatasetModel*, *global_dimension: str*, *global_axis: ArrayLike*, *model_dimension: str*, *model_axis: ArrayLike*) → xr.Dataset

Simulate a dataset model with global megacomplexes.

Parameters

- **dataset_model** (`DatasetModel`) – The dataset model to simulate.
- **global_dimension** (*str*) – The global dimension of the dataset.
- **global_axis** (*ArrayLike*) – The global axis of the dataset.
- **model_dimension** (*str*) – The model dimension of the dataset.

- **model_axis** (*ArrayLike*) – The model axis of the dataset.

Returns
The simulated dataset.

Return type
xr.Dataset

Raises
ValueError – Raised if at least one of the dataset model's global megacomplexes is index dependent.

16.1.12 testing

Testing framework package for glotaran itself and plugins.

Modules

glotaran.testing.plugin_system	Mock functionality for the plugin system.
glotaran.testing.simulated_data	Package containing simulated data for testing and quick demos.

plugin_system

Mock functionality for the plugin system.

Functions

Summary

monkeypatch_plugin_registry	Contextmanager to monkeypatch multiple plugin registries at once.
monkeypatch_plugin_registry_data_io	Monkeypatch the `DataIoInterface` registry.
monkeypatch_plugin_registry_megacomplex	Monkeypatch the `Megacomplex` registry.
monkeypatch_plugin_registry_project_io	Monkeypatch the `ProjectIoInterface` registry.

monkeypatch_plugin_registry

glotaran.testing.plugin_system.**monkeypatch_plugin_registry**(*, *test_megacomplex: MutableMapping[str, type[Megacomplex]] | None = None*, *test_data_io: MutableMapping[str, DataIoInterface] | None = None*, *test_project_io: MutableMapping[str, ProjectIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Contextmanager to monkeypatch multiple plugin registries at once.

Parameters

- **test_megacomplex** (*MutableMapping[str, type[Megacomplex]], optional*) – Registry to to update or replace the `Megacomplex` registry with. , by default None
- **test_data_io** (*MutableMapping[str, DataIoInterface], optional*) – Registry to to update or replace the `DataIoInterface` registry with. , by default None
- **test_project_io** (*MutableMapping[str, ProjectIoInterface], optional*) – Registry to to update or replace the `ProjectIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from the arguments. , by default False

Yields

Generator[None, None, None] – Just keeps all context manager alive

See also:

monkeypatch_plugin_registry_megacomplex, *monkeypatch_plugin_registry_data_io*, *monkeypatch_plugin_registry_project_io*

monkeypatch_plugin_registry_data_io

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_data_io**(*test_data_io: MutableMapping[str, DataIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `DataIoInterface` registry.

Parameters

- **test_data_io** (*MutableMapping[str, DataIoInterface], optional*) – Registry to to update or replace the `DataIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_data_io` , by default False

Yields

Generator[None, None, None] – Just to keep the context alive.

monkeypatch_plugin_registry_megacomplex

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_megacomplex**(*test_megacomplex: MutableMapping[str, type[Megacomplex]] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `Megacomplex` registry.

Parameters

- **test_megacomplex** (*MutableMapping[str, type[Megacomplex]], optional*) – Registry to to update or replace the `Megacomplex` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_megacomplex` , by default False

Yields

Generator[None, None, None] – Just to keep the context alive.

monkeypatch_plugin_registry_project_io

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_project_io**(*test_project_io: MutableMapping[str, ProjectIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `ProjectIoInterface` registry.

Parameters

- **test_project_io** (*MutableMapping[str, ProjectIoInterface], optional*) – Registry to to update or replace the `ProjectIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_data_io` , by default False

Yields

Generator[None, None, None] – Just to keep the context alive.

simulated_data

Package containing simulated data for testing and quick demos.

Modules

glotaran.testing.simulated_data.parallel_spectral_decay	A simple parallel decay for testing purposes.
glotaran.testing.simulated_data.sequential_spectral_decay	A simple sequential decay for testing purposes.
glotaran.testing.simulated_data.shared_decay	Shared variables for simulated decays.

parallel_spectral_decay

A simple parallel decay for testing purposes.

sequential_spectral_decay

A simple sequential decay for testing purposes.

shared_decay

Shared variables for simulated decays.

16.1.13 typing

Glotaran specific typing module.

Modules

glotaran.typing.protocols	Protocol like type definitions.
glotaran.typing.types	Glotaran types module containing commonly used types.

protocols

Protocol like type definitions.

Classes

Summary

FileLoadableProtocol	Protocol class that a file loadable class need adherer to.

FileLoadableProtocol

class glotaran.typing.protocols.**FileLoadableProtocol**(**args*, ***kwargs*)

Bases: Protocol

Protocol class that a file loadable class need adherer to.

Attributes Summary

loader
source_path

loader

FileLoadableProtocol.**loader: Callable[[StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]], *FileLoadableProtocol*]**

source_path

FileLoadableProtocol.**source_path: StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]**

Methods Summary

Methods Documentation

loader: Callable[[StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]], *FileLoadableProtocol*]

source_path: StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]

types

Glotaran types module containing commonly used types.

16.1.14 utils

Glotaran utility function/class package.

Modules

glotaran.utils.attrs_helper	Helper functions for attrs.
glotaran.utils.helpers	Module containing general helper functions.
glotaran.utils.io	Glotaran IO utility module.
glotaran.utils.ipython	Glotaran module with utilities for `ipython` integration (e.g.
glotaran.utils.regex	Glotaran module with regular expression patterns and functions.
glotaran.utils.sanitize	Glotaran module with utilities for sanitation of parsed content.
glotaran.utils.tee	Module containing context manager to capture stdout output and still forward it to stdout.

attrs_helper

Helper functions for attrs.

Functions

Summary

no_default_vals_in_repr	Class decorator to omits attributes from repr that have their default value.

no_default_vals_in_repr

glotaran.utils.attrs_helper.**no_default_vals_in_repr**(*cls*)

Class decorator to omits attributes from repr that have their default value.

Needs to be on top of the `attr.define` decorator. Based on: https://stackoverflow.com/a/47663099/3990615

Parameters
cls – Class decorated with `attr.define`.

Return type
type[cls]

helpers

Module containing general helper functions.

Functions

Summary

nan_or_equal	Compare values which can be nan for equality.

nan_or_equal

glotaran.utils.helpers.**nan_or_equal**(*lhs: Any*, *rhs: Any*) → bool

Compare values which can be nan for equality.

This helper function is needed because `np.nan == np.nan` returns `False`.

Parameters

- **lhs** (*Any*) – Left hand side value.
- **rhs** (*Any*) – Right hand side value.

Returns
Whether or not values are equal.

Return type
bool

io

Glotaran IO utility module.

Functions

Summary

chdir_context	Context manager to change directory to `folder_path`.
create_clp_guide_dataset	Create dataset for clp guidance.
get_script_dir	Get the parent folder a script is executed in.
load_datasets	Load multiple datasets into a mapping (convenience function).
make_path_absolute_if_relative	Get a path as absolute if relative.
relative_posix_path	Ensure that `source_path` is a posix path, relative to `base_path` if defined.
safe_dataframe_fillna	Fill NaN values with `fill_value` if the column exists or do nothing.
safe_dataframe_replace	Replace column values with `replace_value` if the column exists or do nothing.

chdir_context

glotaran.utils.io.**chdir_context**(*folder_path: StrOrPath*) → Generator[Path, None, None]

Context manager to change directory to `folder_path`.

Parameters
folder_path (*StrOrPath*) – Path to change to.

Yields
Generator[Path, None, None] – Resolved path of `folder_path`.

Raises
`ValueError` – If `folder_path` is an existing file.

create_clp_guide_dataset

glotaran.utils.io.**create_clp_guide_dataset**(*result: Result | xr.Dataset*, *clp_label: str*, *dataset_name: str | None = None*) → xr.Dataset

Create dataset for clp guidance.

Parameters

- **result** (*Result | xr.Dataset*) – Optimization result object or dataset, created with pyglotaran>=0.6.0.
- **clp_label** (*str*) – Label of the clp to guide.
- **dataset_name** (*str | None*) – Name of dataset to extract the guide from. Defaults to None.

Returns
DataArray containing the clp guide, with `clp_label` dimension replaced by the model dimensions first value.

Return type
xr.Dataset

Raises

- `ValueError` – If result is an instance of `Result` and `dataset_name` is `None` or not in result.
- `ValueError` – If `clp_labels` is not in result.
- `ValueError` – The result dataset was created with pyglotaran<0.6.0.

Examples

Extracting the clp guide from an optimization result object.

```
from glotaran.io import save_dataset
from glotaran.utils.io import create_clp_guide_dataset

clp_guide = create_clp_guide_dataset(result, "species_1", "dataset_1")
save_dataset(clp_guide, "clp_guide__result_dataset_1__species_1.nc")
```

Extracting the clp guide from a result dataset loaded from file.

```
from glotaran.io import load_dataset
from glotaran.io import save_dataset
from glotaran.utils.io import create_clp_guide_dataset

result_dataset = load_dataset("result_dataset_1.nc")
clp_guide = create_clp_guide_dataset(result_dataset, "species_1")
save_dataset(clp_guide, "clp_guide__result_dataset_1__species_1.nc")
```

get_script_dir

glotaran.utils.io.**get_script_dir**(*, *nesting: int = 0*) → Path

Get the parent folder a script is executed in.

This is a helper function for cross compatibility with jupyter notebooks. In notebooks the global `__file__` variable isn't set, thus we need different means to get the folder a script is defined in, which doesn't change with the current working director the `python interpreter` was called from. :param nesting: Number to go up in the call stack to get to the initially calling function.

This is only needed for library code and not for user code. , by default 0 (direct call)

Returns
Path to the folder the script was resides in.

Return type
Path

load_datasets

glotaran.utils.io.**load_datasets**(*dataset_mappable: str | Path | Dataset | DataArray | Sequence[str | Path | Dataset | DataArray] | Mapping[str, str | Path | Dataset | DataArray]*) → *DatasetMapping*

Load multiple datasets into a mapping (convenience function).

This is used for `file_loadable_field` of a dataset mapping e.g. in `Scheme`

Parameters
dataset_mappable (*DatasetMappable*) – Single dataset/file path to a dataset or sequence or mapping of it.

Returns
Mapping of dataset with string keys, where datasets hare ensured to have the `source_path` attr.

Return type
DatasetMapping

make_path_absolute_if_relative

glotaran.utils.io.**make_path_absolute_if_relative**(*path: Path*) → Path

Get a path as absolute if relative.

Parameters
path (*Path*) – The path to make absolute.

Returns
Either the original path or the path as absolute relative to the script directory.

Return type
Path

relative_posix_path

glotaran.utils.io.**relative_posix_path**(*source_path: StrOrPath*, *base_path: StrOrPath | None = None*) → str

Ensure that `source_path` is a posix path, relative to `base_path` if defined.

For `source_path` to be converted to a relative path it either needs to a an absolute path or `base_path` needs to be a parent directory of `source_path`. On Windows if `source_path` and `base_path` are on different drives, it will return the absolute posix path to the file.

Parameters

- **source_path** (*StrOrPath*) – Path which should be converted to a relative posix path.
- **base_path** (*StrOrPath, optional*) – Base path the resulting path string should be relative to. Defaults to `None`.

Returns
`source_path` as posix path relative to `base_path` if defined.

Return type
str

safe_dataframe_fillna

glotaran.utils.io.**safe_dataframe_fillna**(*df: pd.DataFrame*, *column_name: str*, *fill_value: Any*) → None

Fill NaN values with `fill_value` if the column exists or do nothing.

Parameters

- **df** (*pd.DataFrame*) – DataFrame from which specific column values will be replaced
- **column_name** (*str*) – Name of column of `df` to fill NaNs
- **fill_value** (*Any*) – Value to fill NaNs with

safe_dataframe_replace

glotaran.utils.io.**safe_dataframe_replace**(*df: pd.DataFrame*, *column_name: str*, *to_be_replaced_values: Any*, *replace_value: Any*) → None

Replace column values with `replace_value` if the column exists or do nothing.

If `to_be_replaced_values` is not list or tuple format, convert into list with same `to_be_replaced_values` as element.

Parameters

- **df** (*pd.DataFrame*) – DataFrame from which specific column values will be replaced
- **column_name** (*str*) – Name of column of `df` to replace values for
- **to_be_replaced_values** (*Any*) – Values to be replaced
- **replace_value** (*Any*) – Value to replace `to_be_replaced_values` with

Classes

Summary

DatasetMapping	Wrapper class for a mapping of datasets which can be used for a `file_loadable_field`.

DatasetMapping

class glotaran.utils.io.**DatasetMapping**(*init_map: Mapping[str, Dataset] | None = None*)

Bases: `MutableMapping`

Wrapper class for a mapping of datasets which can be used for a `file_loadable_field`.

Initialize an instance of *DatasetMapping*.

Parameters

init_dict (*dict[str, xr.Dataset] | None*) – Mapping to initially populate the instance. Defaults to `None`.

Attributes Summary

source_path	Map the `source_path` attribute of each dataset to a standalone mapping.

source_path

DatasetMapping.**source_path**

Map the `source_path` attribute of each dataset to a standalone mapping.

Note: When the `source_path` attribute of the dataset gets updated (e.g. by calling `save_dataset` with the default `update_source_path=True`) this value will be updated as well.

Returns
Mapping of the dataset source paths.

Return type
Mapping[str, str]

Methods Summary

clear	
get	
items	
keys	
loader	Loader function utilized by `file_loadable_field`.
pop	If key is not found, d is returned if given, otherwise KeyError is raised.
popitem	as a 2-tuple; but raise KeyError if D is empty.
setdefault	
update	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
values	

clear

DatasetMapping.**clear**() → None. Remove all items from D.

get

DatasetMapping.**get**(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items

DatasetMapping.**items**() → a set-like object providing a view on D's items

keys

DatasetMapping.**keys**() → a set-like object providing a view on D's keys

loader

classmethod DatasetMapping.**loader**(*dataset_mappable: str | Path | Dataset | DataArray | Sequence[str | Path | Dataset | DataArray] | Mapping[str, str | Path | Dataset | DataArray]*) → *DatasetMapping*

Loader function utilized by `file_loadable_field`.

Parameters
dataset_mappable (*DatasetMappable*) – Mapping of datasets to initialize *DatasetMapping*.

Returns
Populated instance of *DatasetMapping*.

Return type
DatasetMapping

pop

DatasetMapping.**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

DatasetMapping.**popitem**() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault

DatasetMapping.**setdefault**(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update

DatasetMapping.**update**([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values

DatasetMapping.**values**() → an object providing a view on D's values

Methods Documentation

clear() → None. Remove all items from D.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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 loader(*dataset_mappable: str | Path | Dataset | DataArray | Sequence[str | Path | Dataset | DataArray] | Mapping[str, str | Path | Dataset | DataArray]*) → *DatasetMapping*

Loader function utilized by `file_loadable_field`.

Parameters
dataset_mappable (*DatasetMappable*) – Mapping of datasets to initialize *DatasetMapping*.

Returns
Populated instance of *DatasetMapping*.

Return type
DatasetMapping

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() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

property source_path

Map the `source_path` attribute of each dataset to a standalone mapping.

Note: When the `source_path` attribute of the dataset gets updated (e.g. by calling `save_dataset` with the default `update_source_path=True`) this value will be updated as well.

Returns
Mapping of the dataset source paths.

Return type
Mapping[str, str]

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

ipython

Glotaran module with utilities for `ipython` integration (e.g. `notebooks`).

Functions

Summary

display_file	Display a file with syntax highlighting `syntax`.

display_file

glotaran.utils.ipython.**display_file**(*path: str | Path*, *, *syntax: str | None = None*) → *MarkdownStr*

Display a file with syntax highlighting `syntax`.

Parameters

- **path** (*StrOrPath*) – Paths to the file
- **syntax** (*str | None*) – Syntax used for syntax highlighting. Defaults to None which means that the syntax is inferred based on the file extension. Pass the value "" to deactivate syntax highlighting.

Returns
File content with syntax highlighting to render in ipython.

Return type
MarkdownStr

Classes

Summary

MarkdownStr	String wrapper class for rich display integration of markdown in ipython.

MarkdownStr

class glotaran.utils.ipython.**MarkdownStr**(*wrapped_str: str*, *, *syntax: str | None = None*)

Bases: UserString

String wrapper class for rich display integration of markdown in ipython.

Initialize string class that is automatically displayed as markdown by `ipython`.

Parameters

- **wrapped_str** (*str*) – String to be wrapped.
- **syntax** (*str*) – Syntax highlighting which should be applied, by default None

Note: Possible syntax highlighting values can e.g. be found here: https://support.codebasehq.com/articles/tips-tricks/syntax-highlighting-in-markdown

Methods Summary

capitalize	
casefold	
center	
count	
encode	
endswith	
expandtabs	
find	
format	
format_map	
index	Raises ValueError if the value is not present.

continues on next page

Table 1 – continued from previous page

isalnum	
isalpha	
isascii	
isdecimal	
isdigit	
isidentifier	
islower	
isnumeric	
isprintable	
isspace	
istitle	
isupper	
join	
ljust	
lower	
lstrip	
maketrans	Return a translation table usable for str.translate().
partition	
removeprefix	
removesuffix	
replace	
rfind	
rindex	
rjust	
rpartition	
rsplit	

continues on next page

Table 1 – continued from previous page

rstrip
split
splitlines
startswith
strip
swapcase
title
translate
upper
zfill

capitalize

MarkdownStr.**capitalize**()

casefold

MarkdownStr.**casefold**()

center

MarkdownStr.**center**(*width*, **args*)

count

MarkdownStr.**count**(*value*) → integer -- return number of occurrences of value

encode

MarkdownStr.**encode**(*encoding='utf-8'*, *errors='strict'*)

endswith

MarkdownStr.**endswith**(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs

MarkdownStr.**expandtabs**(*tabsize=8*)

find

MarkdownStr.**find**(*sub*, *start=0*, *end=9223372036854775807*)

format

MarkdownStr.**format**(**args*, ***kwds*)

format_map

MarkdownStr.**format_map**(*mapping*)

index

MarkdownStr.**index**(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum

MarkdownStr.**isalnum**()

isalpha

MarkdownStr.**isalpha**()

isascii

MarkdownStr.**isascii**()

isdecimal

MarkdownStr.**isdecimal**()

isdigit

MarkdownStr.**isdigit**()

isidentifier

MarkdownStr.**isidentifier**()

islower

MarkdownStr.**islower**()

isnumeric

MarkdownStr.**isnumeric**()

isprintable

MarkdownStr.**isprintable**()

isspace

MarkdownStr.**isspace**()

istitle

MarkdownStr.**istitle**()

isupper

MarkdownStr.**isupper**()

join

MarkdownStr.**join**(*seq*)

ljust

MarkdownStr.**ljust**(*width*, **args*)

lower

MarkdownStr.**lower**()

lstrip

MarkdownStr.**lstrip**(*chars=None*)

maketrans

MarkdownStr.**maketrans**(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition

MarkdownStr.**partition**(*sep*)

removeprefix

MarkdownStr.**removeprefix**(*prefix*, */*)

removesuffix

MarkdownStr.**removesuffix**(*suffix*, */*)

replace

MarkdownStr.**replace**(*old*, *new*, *maxsplit=-1*)

rfind

MarkdownStr.**rfind**(*sub*, *start=0*, *end=9223372036854775807*)

rindex

MarkdownStr.**rindex**(*sub*, *start=0*, *end=9223372036854775807*)

rjust

MarkdownStr.**rjust**(*width*, **args*)

rpartition

MarkdownStr.**rpartition**(*sep*)

rsplit

MarkdownStr.**rsplit**(*sep=None*, *maxsplit=-1*)

rstrip

MarkdownStr.**rstrip**(*chars=None*)

split

MarkdownStr.**split**(*sep=None*, *maxsplit=-1*)

splitlines

MarkdownStr.**splitlines**(*keepends=False*)

startswith

MarkdownStr.**startswith**(*prefix*, *start=0*, *end=9223372036854775807*)

strip

MarkdownStr.**strip**(*chars=None*)

swapcase

MarkdownStr.**swapcase**()

title

MarkdownStr.**title**()

translate

MarkdownStr.**translate**(**args*)

upper

MarkdownStr.**upper**()

zfill

MarkdownStr.**zfill**(*width*)

Methods Documentation

capitalize()

casefold()

center(*width*, **args*)

count(*value*) → integer -- return number of occurrences of value

encode(*encoding='utf-8'*, *errors='strict'*)

endswith(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs(*tabsize=8*)

find(*sub*, *start=0*, *end=9223372036854775807*)

format(**args*, ***kwds*)

format_map(*mapping*)

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum()

isalpha()

isascii()

isdecimal()

isdigit()

isidentifier()

islower()

isnumeric()

isprintable()

isspace()

istitle()

isupper()

join(*seq*)

ljust(*width*, **args*)

lower()

lstrip(*chars=None*)

maketrans(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*)

removeprefix(*prefix*, */*)

removesuffix(*suffix*, */*)

replace(*old*, *new*, *maxsplit=-1*)

rfind(*sub*, *start=0*, *end=9223372036854775807*)

rindex(*sub*, *start=0*, *end=9223372036854775807*)

rjust(*width*, **args*)

rpartition(*sep*)

rsplit(*sep=None*, *maxsplit=-1*)

rstrip(*chars=None*)

split(*sep=None*, *maxsplit=-1*)

splitlines(*keepends=False*)

startswith(*prefix*, *start=0*, *end=9223372036854775807*)

strip(*chars=None*)

swapcase()

title()

translate(**args*)

upper()

zfill(*width*)

regex

Glotaran module with regular expression patterns and functions.

Classes

Summary

RegexPattern	An 'Enum' of (compiled) regular expression patterns (rp).

RegexPattern

class glotaran.utils.regex.**RegexPattern**

Bases: object

An 'Enum' of (compiled) regular expression patterns (rp).

Attributes Summary

elements_in_string_of_list
group
list_with_tuples
number
number_scientific
optimization_stdout
tuple_number
tuple_word
word

elements_in_string_of_list

RegexPattern.**elements_in_string_of_list: Pattern = re.compile('(\\(.+?\\)|[-+.\\d]+)')**

group

RegexPattern.**group: Pattern = re.compile('(\\(.+?\\))')**

list_with_tuples

RegexPattern.**list_with_tuples: Pattern = re.compile('(\\[.+\\(.+\\).+\\])')**

number

RegexPattern.**number: Pattern = re.compile('[\\d.+-]+')**

number_scientific

```
RegexPattern.number_scientific: Pattern =
re.compile('[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)')
```

optimization_stdout

```
RegexPattern.optimization_stdout: Pattern =
re.compile('^\\s+(?P<iteration>\\d+)\\s+(?P<nfev>\\d+)\\s+(?P<cost>\\d\\.\\
d+e[+-]\\d+)(\\s+(?P<cost_reduction>\\d\\.\\d+e[+-]\\d+)\\s+(?P<step_norm>\
\d\\.\\d+e[+-]\\d+)|\\s+)\\s+(?P<optimality>\\d\\.\\d+e[+-]\\,
re.MULTILINE)
```

tuple_number

```
RegexPattern.tuple_number: Pattern =
re.compile('(\\([\\s\\d.+-]+?[,\\s\\d.+-]*?\\))')
```

tuple_word

```
RegexPattern.tuple_word: Pattern =
re.compile('(\\([.\\s\\w\\d]+?[,.\\s\\w\\d]*?\\))')
```

word

```
RegexPattern.word: Pattern = re.compile('[\\w]+')
```

Methods Summary

Methods Documentation

```
elements_in_string_of_list: Pattern = re.compile('(\\(.+?\\)|[-+.\\d]+)')

group: Pattern = re.compile('(\\(.+?\\))')

list_with_tuples: Pattern = re.compile('(\\[.+\\(.+\\).+\\])')

number: Pattern = re.compile('[\\d.+-]+')

number_scientific: Pattern =
re.compile('[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)')
```

```
optimization_stdout: Pattern =
re.compile('^\\s+(?P<iteration>\\d+)\\s+(?P<nfev>\\d+)\\s+(?P<cost>\\d\\.\\
d+e[+-]\\d+)(\\s+(?P<cost_reduction>\\d\\.\\d+e[+-]\\d+)\\s+(?P<step_norm>\
\d\\.\\d+e[+-]\\d+)|\\s+)\\s+(?P<optimality>\\d\\.\\d+e[+-]\\,
re.MULTILINE)

tuple_number: Pattern = re.compile('(\\([\\s\\d.+-]+?[,\\s\\d.+-]*?\\))')

tuple_word: Pattern = re.compile('(\\([.\\s\\w\\d]+?[,.\\s\\w\\d]*?\\))')

word: Pattern = re.compile('[\\w]+')
```

sanitize

Glotaran module with utilities for sanitation of parsed content.

Functions

Summary

convert_scientific_to_float	Convert value to float if it matches scientific notation string.
list_string_to_tuple	Convert a list of strings (representing tuples) to a list of tuples.
pretty_format_numerical	Format value with with at most `decimal_places` decimal places.
sanitize_dict_keys	Sanitize the stringified tuple dict keys in a yaml parsed dict.
sanitize_dict_values	Sanitizes a dict with broken tuples inside modifying it in-place.
sanitize_list_with_broken_tuples	Sanitize a list with 'broken' tuples.
sanitize_parameter_list	Replace in a list strings matching scientific notation with floats.
sanitize_yaml	Sanitize a yaml-returned dict for key or (list) values containing tuples.
sanity_scientific_notation_conversion	Convert scientific notation string values to floats.
string_to_tuple	Convert a string to a tuple if it matches a tuple pattern.

convert_scientific_to_float

glotaran.utils.sanitize.**convert_scientific_to_float**(*value: str*) → float | str

Convert value to float if it matches scientific notation string.

Parameters
value (*str*) – value to convert from string to float if it matches scientific notation

Returns
return float if value was scientific notation string, else turn original value

Return type
float | string

list_string_to_tuple

glotaran.utils.sanitize.**list_string_to_tuple**(*a_list: list[str]*) → list[tuple[float, ...] | tuple[str, ...] | float | str]

Convert a list of strings (representing tuples) to a list of tuples.

Parameters
a_list (*List[str]*) – A list of strings, some of them representing (numbered) tuples

Returns
A list of the (numbered) tuples represted by the incoming a_list

Return type
List[Union[float, str]]

pretty_format_numerical

glotaran.utils.sanitize.**pretty_format_numerical**(*value: float | int*, *decimal_places: int = 1*) → str

Format value with with at most `decimal_places` decimal places.

Used to format values like the t-value.

Parameters

- **value** (*float | int*) – Numerical value to format.
- **decimal_places** (*int*) – Decimal places to display. Defaults to 1

Returns
Pretty formatted version of the value.

Return type
str

sanitize_dict_keys

glotaran.utils.sanitize.**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.utils.sanitize.**sanitize_dict_values**(*d: dict[str, Any] | list[Any]*)

Sanitizes a dict with broken tuples inside modifying it in-place.

Broken tuples are tuples that are turned into strings by the yaml parser. This functions calls *sanitize_list_with_broken_tuples* to glue the broken strings together and then calls list_to_tuple to turn the list with tuple strings back to number tuples.

Parameters
d (*dict*) – A (complex) dict containing (possibly nested) values of broken tuple strings.

sanitize_list_with_broken_tuples

glotaran.utils.sanitize.**sanitize_list_with_broken_tuples**(*mangled_list: list[str | float]*) → list[str]

Sanitize a list with 'broken' tuples.

A list of broken tuples as returned by yaml when parsing tuples. e.g parsing the list of tuples [(3,100), (4,200)] results in a list of str ['(3', '100)', '(4', '200)'] which can be restored to a list with the tuples restored as strings ['(3, 100)', '(4, 200)']

Parameters
mangled_list (*List[Union[str,float]]*) – A list with strings representing tuples broken up by round brackets.

Returns
A list containing the restores tuples (in string form) which can be converted back to numbered tuples using *list_string_to_tuple*

Return type
List[str]

sanitize_parameter_list

glotaran.utils.sanitize.**sanitize_parameter_list**(*parameter_list: list[str | float]*) → list[str | float]

Replace in a list strings matching scientific notation with floats.

Parameters
parameter_list (*list*) – A list of parameters where some elements may be strings like 1E7

Returns
A list where strings matching a scientific number have been converted to float

Return type
list

sanitize_yaml

glotaran.utils.sanitize.**sanitize_yaml**(*d: dict*, *do_keys: bool = True*, *do_values: bool = False*) → dict

Sanitize a yaml-returned dict for key or (list) values containing tuples.

Parameters

- **d** (*dict*) – a dict resulting from parsing a pyglotaran model spec yml file
- **do_keys** (*bool*) – toggle sanitization of dict keys, by default True
- **do_values** (*bool*) – toggle sanitization of dict values, by default False

Returns
a sanitized dict with (broken) string tuples restored as proper tuples

Return type
dict

sanity_scientific_notation_conversion

glotaran.utils.sanitize.**sanity_scientific_notation_conversion**(*d: dict[str, Any] | list[Any]*)

Convert scientific notation string values to floats.

Parameters
d (*dict[str, Any] | list[Any]*) – Iterable which should be checked for scientific notation values.

string_to_tuple

glotaran.utils.sanitize.**string_to_tuple**(*tuple_str: str*, *from_list=False*) → tuple[float, ...] | tuple[str, ...] | float | str

Convert a string to a tuple if it matches a tuple pattern.

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
tuple[float], tuple[str], float, str

tee

Module containing context manager to capture stdout output and still forward it to stdout.

Classes

Summary

TeeContext	Context manager that allows to work with string written to stdout.

TeeContext

class glotaran.utils.tee.**TeeContext**

Bases: object

Context manager that allows to work with string written to stdout.

This context manager behaves similar to the `tee` shell command. https://linuxize.com/post/linux-tee-command

Create new `StringIO` buffer and save reference to original `sys.stdout`.

Methods Summary

flush	Flush values in the buffer.
read	Return values written to buffer.
write	Write to buffer and original `sys.stdout`.

flush

TeeContext.**flush**() → None

Flush values in the buffer.

read

TeeContext.**read**() → str

Return values written to buffer.

Returns
Text written to buffer.

Return type
str

write

TeeContext.**write**(*data: str*) → None

Write to buffer and original `sys.stdout`.

Parameters
data (*str*) – String to write to stdout and buffer.

Methods Documentation

flush() → None

Flush values in the buffer.

read() → str

Return values written to buffer.

Returns
Text written to buffer.

Return type
str

write(*data: str*) → None

Write to buffer and original `sys.stdout`.

Parameters
data (*str*) – String to write to stdout and buffer.

CHAPTER

SEVENTEEN

PLUGIN DEVELOPMENT

If you don't find the plugin that fits your needs you can always write your own. This sections will explain you how and what you need to know.

In time we will also provide you with a cookiecutter template, to kickstart your new plugin for publishing as a package on PyPi.

The following section was generated from `docs/source/notebooks/plugin_system/plugin_howto_write_a_io_plugin.ipynb`

17.1 How to Write your own Io plugin

There are all kinds of different data formats, so it is quite likely that your experimental setup uses a format which isn't yet supported by a `glotaran` plugin and want to write your own `DataIo` plugin to support this format.

Since `json` is very common format (admittedly not for data, but in general) and python has builtin support for it we will use it as an example.

First let's have a look which `DataIo` plugins are already installed and which functions they support.

[1]:
```
from glotaran.io import data_io_plugin_table
```

[2]:
```
data_io_plugin_table()
```

[2]:

Format name	load_dataset	save_dataset
ascii	*	*
nc	*	*
sdt	*	/

Looks like there isn't a `json` plugin installed yet, but maybe someone else did already write one, so have a look at the `3rd party plugins` list in the user docsumentation <https://pyglotaran.readthedocs.io/en/latest/user_documentation/using_plugins.html>`__ before you start writing your own plugin.

For the sake of the example, we will write our `json` plugin even if there already exists one by the time you read this.

First you need to import all needed libraries and functions.

- `from __future__ import annotations`: needed to write python 3.10 typing syntax (`|`), even with a lower python version
- `json`,`xarray`: Needed for reading and writing itself
- `DataIoInterface`: needed to subclass from, this way you get the proper type and especially signature checking
- `register_data_io`: registers the DataIo plugin under the given `format_name`s

[3]:
```
from __future__ import annotations

import json

import xarray as xr

from glotaran.io.interface import DataIoInterface
from glotaran.plugin_system.data_io_registration import register_data_io
```

`DataIoInterface` has two methods we could implement `load_dataset` and `save_dataset`, which are used by the identically named functions in `glotaran.io`.

We will just implement both for our example to be complete. the quickest way to get started is to just copy over the code from `DataIoInterface` which already has the right signatures and some boilerplate docstrings, for the method arguments.

If the default arguments aren't enough for your plugin and you need your methods to have additional option, you can just add those. Note the `*` between `file_name` and `my_extra_option`, this tell python that `my_extra_option` is an keyword only argument and `mypy <https://github.com/python/mypy>`__ won't raise an `[override]` type error for changing the signature of the method. To help others who might use your plugin and your future self, it is good practice to documents what each parameter does in the methods docstring, which will be accessed by the help function.

Finally add the `@register_data_io` with the `format_name`'s you want to register the plugin to, in our case `json` and `my_json`.

Pro tip: You don't need to implement the whole functionality inside of the method itself,

[4]:
```
@register_data_io(["json", "my_json"])
class JsonDataIo(DataIoInterface):
    """My new shiny glotaran plugin for json data io"""

    def load_dataset(
        self, file_name: str, *, my_extra_option: str = None
    ) -> xr.Dataset | xr.DataArray:
        """Read json data to xarray.Dataset


        Parameters
        ----------
        file_name : str
            File containing the data.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option loading json: {my_extra_option}")

        with open(file_name) as json_file:
            data_dict = json.load(json_file)
        return xr.Dataset.from_dict(data_dict)

    def save_dataset(
        self, dataset: xr.Dataset | xr.DataArray, file_name: str, *, my_extra_option=None
    ):
        """Write xarray.Dataset to a json file
```

(continues on next page)

(continued from previous page)

```
        Parameters
        ----------
        dataset : xr.Dataset
            Dataset to be saved to file.
        file_name : str
            File to write the result data to.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option for writing json: {my_extra_option}")

        data_dict = dataset.to_dict()
        with open(file_name, "w") as json_file:
            json.dump(data_dict, json_file)
```

Let's verify that our new plugin was registered successfully under the `format_names` `json` and `my_json`.

[5]:
```
data_io_plugin_table()
```

[5]:

Format name	load_dataset	save_dataset
ascii	*	*
json	*	*
my_json	*	*
nc	*	*
sdt	*	/

Now let's use the example data from the quickstart to test the reading and writing capabilities of our plugin.

[6]:
```
from glotaran.io import load_dataset
from glotaran.io import save_dataset
from glotaran.testing.simulated_data.sequential_spectral_decay import DATASET as dataset
```

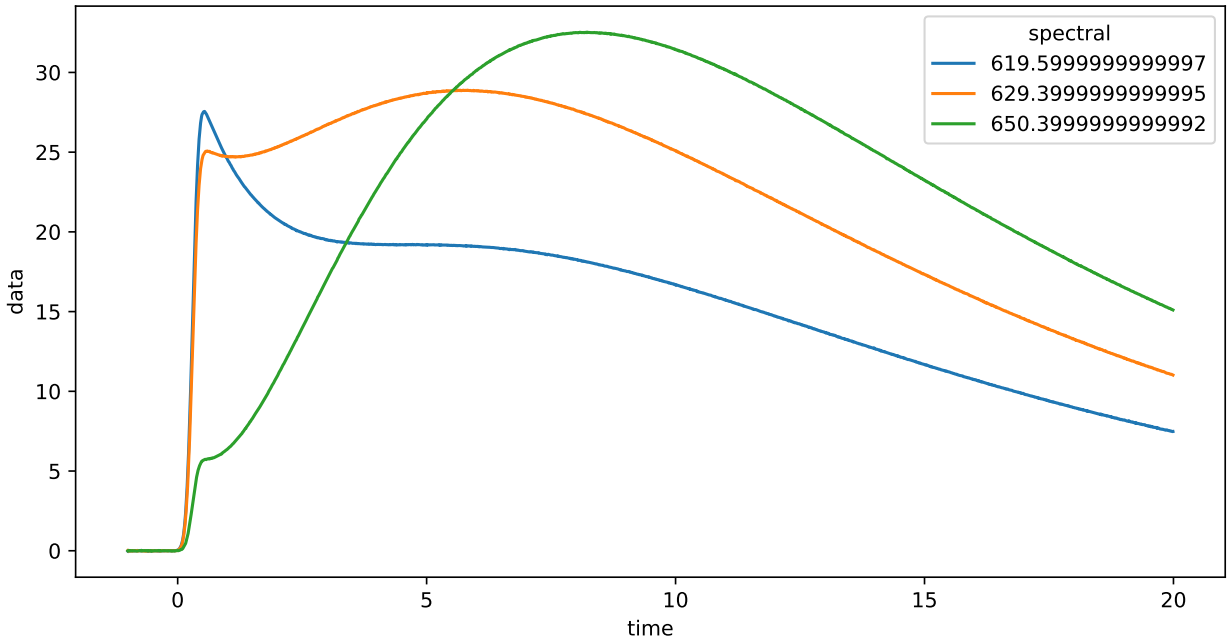
[7]:
```
dataset
```

[7]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
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 -0.01251 0.02317 0.005785 ... 2.549 2.31
Attributes:
    source_path:  dataset_1.nc
```

To get a feeling for our data, let's plot some traces.

[8]:
```
plot_data = dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7fb700da5120>,
 <matplotlib.lines.Line2D at 0x7fb700da51e0>,
 <matplotlib.lines.Line2D at 0x7fb700da51b0>]
```



Since we want to see a difference of our saved and loaded data, we divide the amplitudes by 2 for no reason.

[9]:
```
dataset["data"] = dataset.data / 2
```

Now that we changed the data, let's write them to a file.

But in which order were the arguments again? And are there any additional option?

Good thing we documented our new plugin, so we can just lookup the help.

[10]:
```
from glotaran.io import show_data_io_method_help

show_data_io_method_help("json", "save_dataset")
```

```
Help on method save_dataset in module __main__:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'str', *, my_extra_
↪option=None) method of __main__.JsonDataIo instance
    Write xarray.Dataset to a json file

    Parameters
    ----------
    dataset : xr.Dataset
        Dataset to be saved to file.
    file_name : str
        File to write the result data to.
    my_extra_option: str
        This argument is only for demonstration
```

Note that the **function** `save_dataset` has additional arguments:

- `format_name`: overwrites the inferred plugin selection
- `allow_overwrite`: Allows to overwrite existing files **(USE WITH CAUTION!!!)**

[11]:
```
help(save_dataset)
```

```
Help on function save_dataset in module glotaran.plugin_system.data_io_registration:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'StrOrPath', format_name:
↪'str | None' = None, *, data_filters: 'list[str] | None' = None, allow_overwrite: 'bool
↪' = False, update_source_path: 'bool' = True, **kwargs: 'Any') -> 'None'
    Save data from :xarraydoc:`Dataset` or :xarraydoc:`DataArray` to a file.

    Parameters
    ----------
    dataset : xr.Dataset | xr.DataArray
        Data to be written to file.
    file_name : StrOrPath
        File to write the data to.
    format_name : str
        Format the file should be in, if not provided it will be inferred from the file␣
↪extension.
    data_filters : list[str] | None
        Optional list of items in the dataset to be saved.
    allow_overwrite : bool
        Whether or not to allow overwriting existing files, by default False
    update_source_path: bool
        Whether or not to update the ``source_path`` attribute to ``file_name`` when␣
↪saving.
        by default True
    **kwargs : Any
        Additional keyword arguments passes to the ``write_dataset`` implementation
        of the data io plugin. If you aren't sure about those use ``get_datasaver``
        to get the implementation with the proper help and autocomplete.
```

Since this is just an example and we don't overwrite important data we will use `allow_overwrite=True`. Also it makes writing this documentation easier, not having to manually delete the test file each time you run the cell.

[12]:
```
save_dataset(
    dataset, "half_intensity.json", allow_overwrite=True, my_extra_option="just as an␣
↪example"
)
```

```
Using my extra option for writing json: just as an example
```

Now let's test our data loading functionality.

[13]:
```
reloaded_data = load_dataset("half_intensity.json", my_extra_option="just as an example")
reloaded_data
```

```
Using my extra option loading json: just as an example
```

[13]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
```

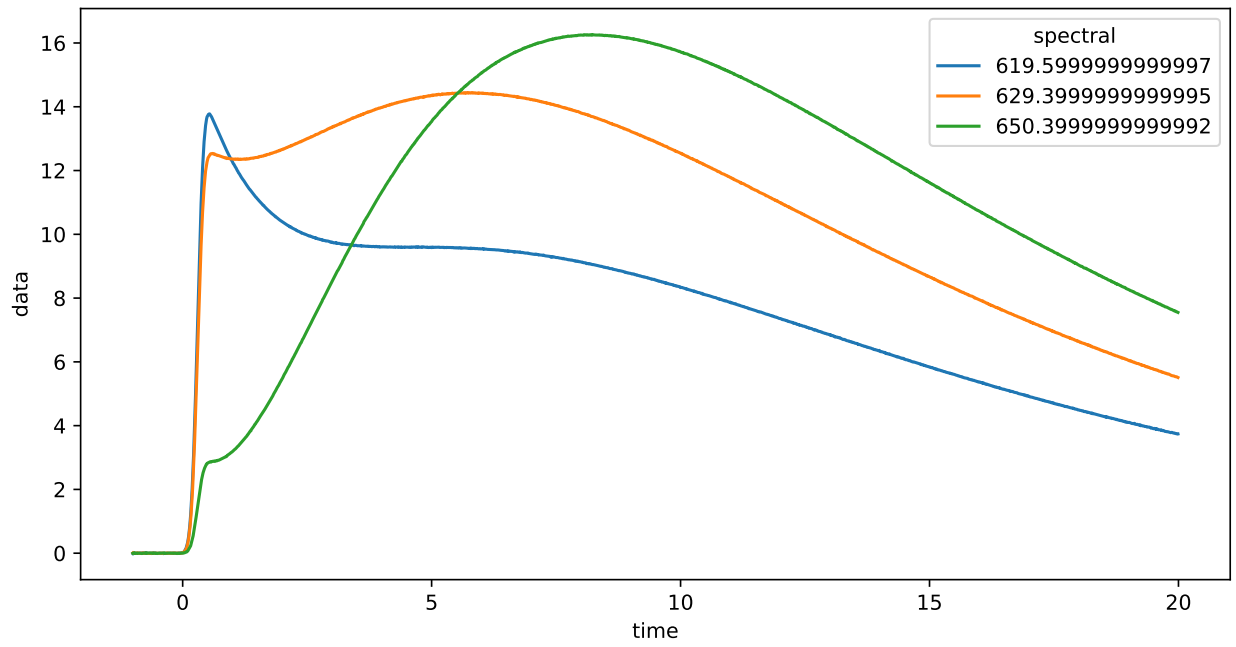
(continues on next page)

(continued from previous page)

```
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 -0.006257 0.01158 ... 1.274 1.155
Attributes:
    source_path:  half_intensity.json
    loader:       <function load_dataset at 0x7fb7132ec790>
```

[14]:
```
reloaded_plot_data = reloaded_data.data.sel(spectral=[620, 630, 650], method="nearest")
reloaded_plot_data.plot.line(x="time", aspect=2, size=5)
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7fb6fe9242e0>,
 <matplotlib.lines.Line2D at 0x7fb6fe924370>,
 <matplotlib.lines.Line2D at 0x7fb6fe9243a0>]
```



Since this looks like the above plot, but with half the amplitudes, so writing and reading our data worked as we hoped it would.

Writing a `ProjectIo` plugin words analogous:

	DataIo plugin	ProjectIo plugin
Register function	glotaran.plugin_system.data_io_registration.register_data_io	glotaran.plugin_system.project_io_registration.register_project_io
Baseclass	glotaran.io.interface.DataIoInterface	glotaran.io.interface.DataIoInterface
Possible methods	load_dataset , save_dataset	load_model , save_model , load_parameters , save_parameters , load_scheme , save_scheme , load_result , save_result

Of course you don't have to implement all methods (sometimes that doesn't even make sense), but only the ones you need.

Last but not least:

Chances are that if you need a plugin someone else does too, so it would awesome if you would publish it open source, so the wheel isn't reinvented over and over again.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

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

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

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z