Eddington Documentation

Sagi Shadur

Feb 24, 2021

Contents

1 Installation **3**
1.1 About Us 3
1.2 Community 7
1.3 API 9
1.4 Tutorials 30

Index **35**

Eddington is a user-friendly data fitting platform for common uses, based on open-source libraries such as *numpy*, *scipy*, and *matplotlib*. *Eddington* can be integrated with python code as a library, can be used as a CLI and has an easy to use GUI.

Note: This is the Eddington python library documentation. In order to read about the Eddington user interface go here.

CHAPTER 1

Installation

In order to install eddington use:

```
pip install eddington
```

1.1 About Us

1.1.1 What's Eddington?

Eddington is a user-friendly data fitting platform for common uses, based on open-source libraries such as *numpy*, *scipy*, and *matplotlib*.

Core abilities

With Eddington you can:

- Fit data according to fitting functions.
- Use a vast list of supported *out-of-the-box fitting functions*.
- Plot results into useful figures to evaluate your results.
- Filter your data in order to get best results.
- Save your results into files.

Usage

Eddington can be operated in 3 major ways:

As a Library

With just 4 lines of code you can fit your data easily and surely:

```
from eddington import FittingData, fit

from eddington import linear  # Import the needed fitting function

data = FittingData.read_from_csv("/path/to/data.csv")  # Read the data from a file

result = fit(data, linear)  # Fit the data

print(result)  # Print the result or export to file
```

The Eddington library is simple, intuitive and easy to use for all developers with basic knowledge of python. Install the Eddington library with `pip install eddington`.

As a Command-Line Interface (CLI)

Install Eddington-CLI using `pip install eddington-cli` and run Eddington via the command line:

```
>> eddington fit hyperbolic --data=/path/to/data.csv
Fitting hyperbolic (a[0] / (x + a[1]) + a[2])
Results:
========

Initial parameters' values:
        1.0 1.0 1.0
Fitted parameters' values:
        a[0] = 245.670 ± 8.530 (3.472% error)
        a[1] = -16.329 ± 0.1228 (0.7518% error)
        a[2] = 14.515 ± 0.5415 (3.731% error)
Fitted parameters covariance:
[[ 1.834e+02  2.579e+00 -1.145e+01]
 [ 2.579e+00  3.799e-02 -1.560e-01]
 [-1.145e+01 -1.560e-01  7.390e-01]]
Chi squared: 3.174
Degrees of freedom: 8
Chi squared reduced: 0.3968
P-probability: 0.9230
```

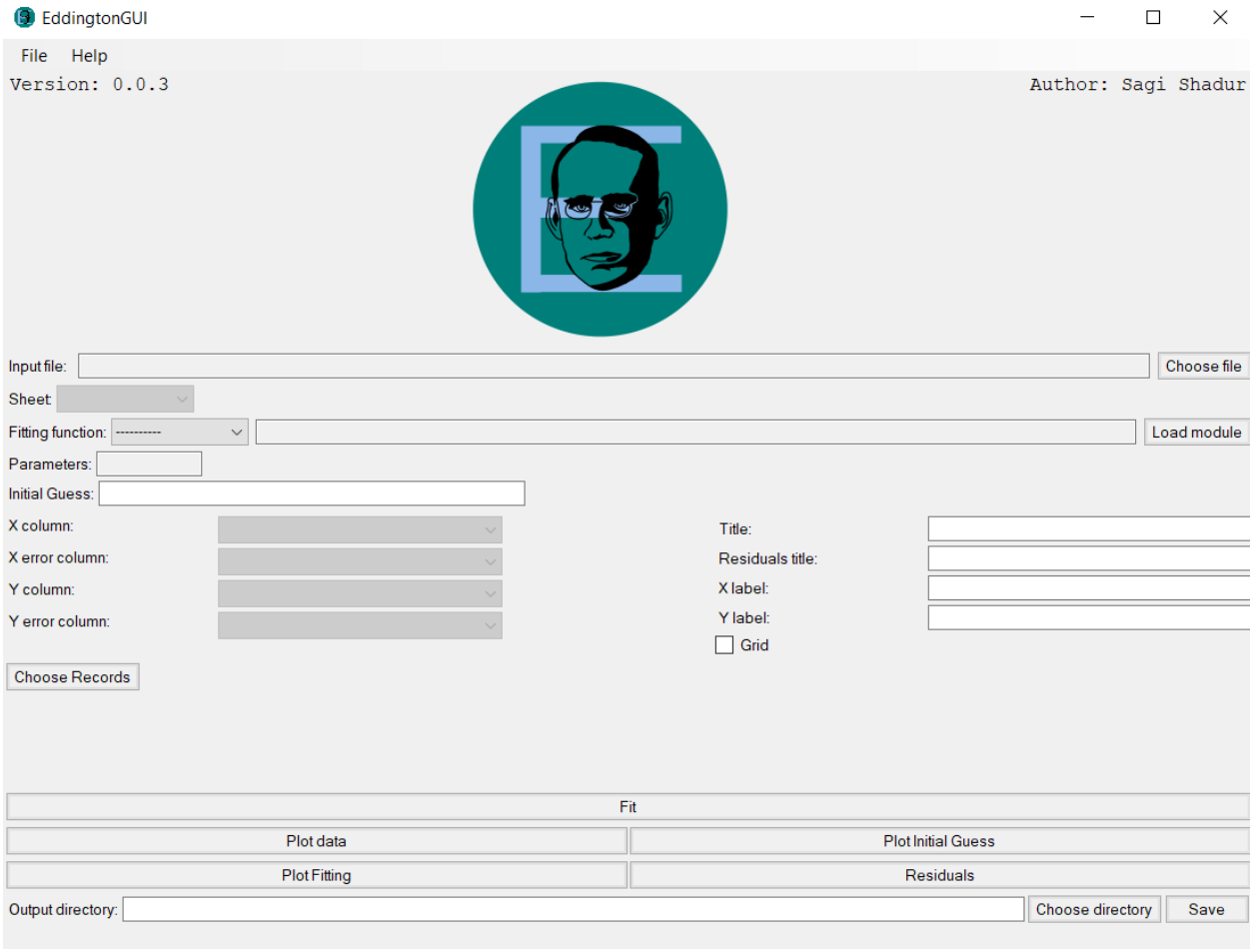
As a Graphical User Interface (GUI)

Install Eddington-GUI from here and run Eddington via a user-friendly GUI:

1.1.2 Who's Eddington?

In Transit (for Arthur Eddington), by Neil Gaiman

...
Light bends around us. So we run,
as gravity reclassifies


EddingtonGUI
File
Help
Version: 0.0.3
Author: Sagi Shadur
Input file:
Choose file
Sheet:
Fitting function:

Load module
Parameters:
Initial Guess:
X column:
X error column:
Y column:
Y error column:
Title:
Residuals title:
X label:
Y label:
Grid
Choose Records
Fit
Plot data
Plot Initial Guess
Plot Fitting
Residuals
Output directory:
Choose directory
Save

the stars we saw behind the sun.

To see the world beyond the skies,
to know the mind behind the eyes,
To find the many in the one
he showed us stars behind the sun.
. . .

Sir Arthur Eddington (28 December 1882 – 22 November 1944) was an English astronomer, physicist, and mathematician. His paper "The Internal Constitution of the Stars" (1920) predicted the discovery of nuclear fusion inside stars.

During the solar eclipse of 1919 he conducted the first experiment to demonstrate Einstein's theory of general relativity. Even though the reliability of the experiment is questionable, it was ground breaking. It was the first time that an English physicist agreed to even consider a theory that was published by a German physicist during World War I – a theory so radical, which contradicts another great English physicist, Sir Isaac Newton.

It was Eddington's dedication to the real, free scientific mind, one which isn't bound by geographical or political borders, that helped Einstein's theory make its first steps toward global consensus. Today, Einstein's theory of general relativity has been proven hundreds of times, based on thousands of observations, all which started by Eddington's first attempt.

Eddington died from cancer at the age of 61. He had no wife and no children.

For his dedication to the scientific method, and his dedicated quest towards the world's hidden truths, we decided to honour his name by naming our data fitting platform after him.

1.1.3 Why Should I Use Eddington?

- It's high-level so you don't need to write many lines of code in order to get a decent fit.
- It's open-source and free to use.
- It's based on other state-of-the-art scientific python libraries such as *numpy*, *scipy* and *matplotlib*.
- It supports a vast list of out-of-the-box fitting functions.
- It can read and write data from many of the standard data files formats such as CSV, Json and Excel.
- It's fast and accurate.
- You can use Eddington in your most convenient way: as a python library, as a CLI or as a GUI.
- It's expendable: You can build upon it to write data manipulation functions, new fitting functions and etc.

Is There A Reason I Won't Want to Use Eddington?

You may want to consider other platforms if:

- You prefer Matlab over Python. In that case, Eddington won't help you.
- You want to fit your data in an unorthodox way.
- You want to "put your hands in the mud" and use a lower-level fitting library.
- You don't like English scientists.

1.2 Community

1.2.1 Acknowledgement

Creator

- Sagi Shadur

Logo Design

- Rotem Shadur

Professional Advisory

- Prof. Halina Abramovich
- Aviv Karnieli
- Avia Raviv
- Itai Silber
- Sivan Trajtenberg Mills

Contributors

- Assaf Zohar
- Ohad Nir
- Yuval Bardugo

Special Thanks

- Adi Shadur for linguistic advisory.
- The warm and supportive members of the open-source community, who share their code and their hearts with the rest of the world.
- The great minds behind *numpy*, *scipy* and *matplotlib*, which without them this project could never come to exist.

1.2.2 Contribution Guide

Thank you for choosing to spend your time contributing to the Eddington platform. Contributors are the living, beating heart of any open-source project, and we really appreciate each and every contribution made to our code-base, big or small.

Here is a relatively short guide on how to contribute code to Eddington. Please follow the next steps carefully when making a pull request.

Step 1 – Writing Your Code

Like any other open-source project in Github, contribution to Eddington is done via a pull-request (PR). Fork the desired repository, open a feature branch, and write your code in it.

When writing code, please pay attention that you:

1. Make sure your *master* branch is up-to-date with the latest changes in the Eddington platform, and make your feature branch based upon it. This will help you avoid merge conflicts.
2. Write your code clearly, with self-explainable variables, functions and classes.
3. Reuse existing code when possible.
4. Document new functions, classes, and modules (especially if they're public).

The code reviews you'll receive would often address the following guidelines, as well as any existing design issues.

Step 2 – Testing Your Code

In the Eddington platform, we believe in 100% test coverage, and we enforce it throughout our repositories! If you add new functionalities or change an existing functionality, you must test the new ability with a unit test (or better yet – **unit tests**).

We use pytest as our testing platform. Some of our tests use the pytest-case and pytest-mock libraries. Feel free to use those libraries as well as other testing libraries whenever needed.

In order to run the unit tests and see the code coverage, you should use tox:

1. Install *tox* with `pip install tox`
2. Go to the main repository directory and run `tox -e py`
3. This will run the unit tests and show the coverage report. Make sure the code passes and all lines are covered

We will never compromise on code coverage and/or extensive unit testing.

Step 3 – Cleaning Your Code

Writing a working code can sometimes be really hard, but writing a **clean** code is always harder.

Here on the Eddington platform we believe that code should be clean, and we want to make sure that writing clean code is as easy as possible. We do that by using automatic tools that would help you achieve that along the way.

We use state-of-the-art static code analysis tools such as *black*, *flake8*, *pylint*, *mypy*, *pydocstyle*, etc. Statue is orchestrating all these tools by running each of them on all of our code-base.

In order to use *Statue*, follow the next steps:

1. Run `pip install statue`.
2. Run `statue install`. If needed, this command will install missing packages.
3. Go to the main repository directory and run `statue run --context format`. This will change your code to fit styling guidelines. Save the changes in a commit or append them to an existing commit.
4. Run `statue run` again (now without any arguments) and it will check if there are any issues that it wasn't able to solve on its own. If there are any errors, fix them.
5. Save all changes in a commit or append them to an existing commit.

You may find some of the errors presented by those tools tedious or irrelevant, but rest assured that we take those errors seriously.

If you think that in a specific line an error should be ignored (using `# noqa` or `# pylint: disable` for example), please make sure that this skip is justified before applying it.

Step 4 – Re-running everything

Go to the main repository directory and run `tox` (without any arguments). This will re-run all the different checks we use in Eddington.

Make sure that everything is checked out before continuing to the next step.

Step 5 – Adding Yourself to the Acknowledgment File

We acknowledge each and every one of our contributors in *docs/acknowledgment.rst*. Add your name to the contributors file, keeping alphabetical order.

Step 6 – Receiving a Code Review and Merging

Push the branch and open a PR. We will make our best efforts to review your PR as soon as possible.

Once you receive a code-review, address the issues presented to you by changing the code or commenting back. Once all the issues are resolved, your PR will be merged to master!

1.3 API

1.3.1 Fitting Data

class eddington.fitting_data.**FittingData**(*data: Union[collections.OrderedDict, Dict[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8116110>]], x_column: Union[str, int, None] = None, xerr_column: Union[str, int, None] = None, y_column: Union[str, int, None] = None, yerr_column: Union[str, int, None] = None, search: bool = True*)

Fitting data class.

Constructor.

Parameters

- **data** (dict or OrderedDict from str to numpy.ndarray) – Dictionary from a column name to its values
- **x_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **xerr_column** (str or numpy.ndarray) – Indicates which column should be used as the x error parameter
- **y_column** (str or numpy.ndarray) – Indicates which column should be used as the y parameter

- **yerr_column** (str or numpy.ndarray) – Indicates which column should be used as the y error parameter
- **search** (*bool*) – Search for a column if it wasn't explicitly provided in the constructor.

Raises **FittingDataColumnsLengthError** – Raised if not all columns have the same length

all_columns
Property of columns list.

Returns list of all columns

Return type List[str]

all_records
Get all records in data as a list.

Returns List of all records

Return type List[List[Any]]

all_selected() → bool
Checks whether all records are selected.

Returns True if all records are selected, False otherwise.

Return type bool

cell_data(*column_name: str*, *index: int*) → float
Get the data of a column.

Parameters

- **column_name** (*str*) – The header name of the desired cell.
- **index** (*int*) – The index of the desired cell.

Returns The value of the cell

Return type float

column_data(*column_name: Optional[str], only_selected: bool = True*) → Optional[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8127150>]
Get the data of a column.

Parameters

- **column_name** (*str*) – The header name of the desired column. if None, return None
- **only_selected** (*bool*) – If true, return only values selected records. otherwise, Return values of all records.

Returns The data of the given column

Return type numpy.ndarray or None

column_domain(*column_name: Optional[str], only_selected: bool = True*) → eddington.interval.Interval
Get the smallest interval containing the values in a column.

Parameters

- **column_name** (*str*) – The desired column name.
- **only_selected** (*bool*) – If true, return only values selected records. otherwise, Return values of all records.

Returns Minimal interval containing column values

Return type Interval

copy(*only_selected_columns: bool = False*, *only_selected_records: bool = False*)
Make a copy of self.

Parameters

- **only_selected_columns** (*bool*) – If true, copy only columns which are used as x, x error, y and y error columns. Otherwise, copy all columns
- **only_selected_records** (*bool*) – If true, copy only selected records. Otherwise, copy all records.

Returns a copy of this fitting data.

Return type Fitting Data

data
Data matrix.

Returns The actual raw data

Return type OrderedDict

is_selected(*index*)
Checks if a record is selected or not.

Parameters **index** (*int*) – index of the desired record **starting from 1**.

Returns True if record is selected, otherwise False.

Return type bool

non_selected() → bool
Checks whether no record has been selected.

Returns True if no record has been selected, False otherwise.

Return type bool

number_of_columns
Number of columns.

Returns Number of columns

Return type int

number_of_records
Number of records.

Returns Number of records

Return type int

classmethod read_from_csv(*filepath: Union[str, pathlib.Path]*, *x_column: Union[str, int, None] = None*, *xerr_column: Union[str, int, None] = None*, *y_column: Union[str, int, None] = None*, *yerr_column: Union[str, int, None] = None*, *search: bool = True*)
Read *FittingData* from csv file.

Parameters

- **filepath** – str or Path. Path to location of csv file
- **x_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter

- **xerr_column** (str or numpy.ndarray) – Indicates which column should be used as the x error parameter
- **y_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **yerr_column** (str or numpy.ndarray) – Indicates which column should be used as the y error parameter
- **search** (*bool*) – Search for a column if it wasn't explicitly provided.

Returns *FittingData* read from the csv file.

classmethod read_from_excel(*filepath: Union[str, pathlib.Path], sheet: str, x_column: Union[str, int, None] = None, xerr_column: Union[str, int, None] = None, y_column: Union[str, int, None] = None, yerr_column: Union[str, int, None] = None, search: bool = True*)

Read *FittingData* from excel file.

Parameters

- **filepath** – str or Path. Path to location of excel file
- **sheet** – str. The name of the sheet to extract the data from.
- **x_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **xerr_column** (str or numpy.ndarray) – Indicates which column should be used as the x error parameter
- **y_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **yerr_column** (str or numpy.ndarray) – Indicates which column should be used as the y error parameter
- **search** (*bool*) – Search for a column if it wasn't explicitly provided.

Returns *FittingData* read from the excel file.

Raises **FittingDataError** – Raised when the given sheet do not exist in excel file.

classmethod read_from_json(*filepath: Union[str, pathlib.Path], x_column: Union[str, int, None] = None, xerr_column: Union[str, int, None] = None, y_column: Union[str, int, None] = None, yerr_column: Union[str, int, None] = None, search: bool = True*)

Read *FittingData* from json file.

Parameters

- **filepath** – str or Path. Path to location of csv file
- **x_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **xerr_column** (str or numpy.ndarray) – Indicates which column should be used as the x error parameter
- **y_column** (str or numpy.ndarray) – Indicates which column should be used as the x parameter
- **yerr_column** (str or numpy.ndarray) – Indicates which column should be used as the y error parameter

- **search** (*bool*) – Search for a column if it wasn't explicitly provided.

Returns *FittingData* read from the json file.

record_data (*index: int*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8127250>
Get the data of a column.

Parameters **index** (*int*) – The index of the desired record.

Returns The record

Return type numpy.ndarray

records
Get all selected records in data as a list.

Returns records list

records_indices
Property of selected indices.

Returns List of booleans indicating which records are selected.

Return type List[bool]

residuals (*fit_func, a: Union[List[float], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8116f90>]*) → eddington.fitting_data.FittingData
Creates residuals *FittingData* objects.

Parameters

- **fit_func** (FittingFunction) – FittingFunction to evaluate with the fit data
- **a** (numpy.ndarray) – the parameters of the given fitting function

Returns residuals *FittingData*

Raises **FittingDataError** – Raised when missing x or y column

save_csv (*output_directory: Union[str, pathlib.Path], name: str = 'fitting_data'*)
Save *FittingData* to csv file.

Parameters

- **output_directory** (Path or str) – Path to the directory for the new excel file to be saved.
- **name** (*str*) – Optional. The name of the file, without the .csv suffix. "fitting_data" by default.

save_excel (*output_directory: Union[str, pathlib.Path], name: str = 'fitting_data', sheet: Optional[str] = None*)
Save *FittingData* to xlsx file.

Parameters

- **output_directory** (Path or str) – Path to the directory for the new excel file to be saved.
- **name** (*str*) – Optional. The name of the file, without the .xlsx suffix. "fitting_data" by default.
- **sheet** (*str*) – Optional. Name of the sheet that the data will be saved to.

select_all_records ()
Select all records to be used in fitting.

select_by_domains(*xmin: Optional[float] = None*, *xmax: Optional[float] = None*, *ymin: Optional[float] = None*, *ymax: Optional[float] = None*, *update_selected: bool = False*) → None

Select records by limiting y values.

Parameters

- **xmin** (*float*) – Optional. Minimum value for x. If none, will not consider lower bound for x values
- **xmax** (*float*) – Optional. Maximum value for x. If none, will not consider upper bound for x values
- **ymin** (*float*) – Optional. Minimum value for y. If none, will not consider lower bound for y values
- **ymax** (*float*) – Optional. Maximum value for y. If none, will not consider upper bound for y values
- **update_selected** (*bool*) – If true, combine with records which have already been selected. If false, select from all records

select_by_x_domain(*xmin: Optional[float] = None*, *xmax: Optional[float] = None*, *update_selected: bool = False*) → None

Select records by limiting x values.

Parameters

- **xmin** (*float*) – Optional. Minimum value for x. If none, will not consider lower bound for x values
- **xmax** (*float*) – Optional. Maximum value for x. If none, will not consider upper bound for x values
- **update_selected** (*bool*) – If true, combine with records which have already been selected. If false, select from all records

select_by_y_domain(*ymin: Optional[float] = None*, *ymax: Optional[float] = None*, *update_selected: bool = False*) → None

Select records by limiting y values.

Parameters

- **ymin** (*float*) – Optional. Minimum value for y. If none, will not consider lower bound for y values
- **ymax** (*float*) – Optional. Maximum value for y. If none, will not consider upper bound for y values
- **update_selected** (*bool*) – If true, combine with records which have already been selected. If false, select from all records

select_record(*index: int*)

Select a record to be used in fitting.

Parameters **index** (*int*) – index of the desired record **starting from 1**.

set_cell(*column_name: str*, *index: int*, *value: float*)

Set new value to a cell.

Parameters

- **column_name** (*str*) – The column name
- **index** (*int*) – The number of the record to set, starting from 1

- **value** (*float*) – The new value to set for the cell

Raises **FittingDataSetError** – Raised when trying to set a cell with non number value

set_header(*old_column*, *new_column*)
Rename header.

Parameters

- **old_column** (*str*) – The old columns name
- **new_column** (*str*) – The new value to set for the header

Raises **FittingDataSetError** – Raised when trying to set a header which is empty or already been set.

statistics(*column_name: str*) → Optional[eddington.statistics.Statistics]
Get statistics of the values in a column.

Parameters **column_name** (*str*) – The column name to get statistics of

Returns Statistics of the given column

Return type Statistics

Raises **FittingDataColumnExistenceError** – Raised when unknown column name is given.

statistics_map
Return updated statistics map.

Returns Statistics map of the data

Return type Statistics

unselect_all_records()
Unselect all records from being used in fitting.

unselect_record(*index: int*)
Unselect a record to be used in fitting.

Parameters **index** (*int*) – index of the desired record **starting from 1**.

used_columns
Dictionary of columns in use.

Returns columns used dictionary

Return type Columns

x
Property of the x values.

Returns values of the x column

Return type np.ndarray

x_column
Name of the x column.

Returns The name of the x error column

Return type str

x_domain
Minimal interval containing the values of the x column.

Returns x values domain.

Return type Interval

x_index
Index of the x column.

Returns Index of the x column

Return type int

xerr
Property of the x error values.

Returns values of the x error column

Return type np.ndarray

xerr_column
Name of the x error column.

Returns The name of the x error column

Return type str

xerr_index
Index of the x error column.

Returns Index of the x error column

Return type int

y
Property of the y values.

Returns values of the y column

Return type np.ndarray

y_column
Name of the y column.

Returns The name of the y column

Return type str

y_domain
Minimal interval containing the values of the y column.

Returns y values domain.

Return type Interval

y_index
Index of the y column.

Returns Index of the y column

Return type int

yerr
Property of the y error values.

Returns values of the y error column

Return type np.ndarray

yerr_column
Name of the y error column.

Returns The name of the y error column

Return type str

yerr_index
Index of the y error column.

Returns Index of the y error column

Return type int

1.3.2 Fitting Function

FittingFunction Class

class eddington.fitting_function_class.**FittingFunction**(*fit_func: Callable*, *n: int*, *name: str*, *syntax: Optional[str] = None*, *a_derivative: Optional[Callable] = None*, *x_derivative: Optional[Callable] = None*, *save: dataclasses.InitVar = True*)

Fitting function class.

This class wraps up a callable which gets 2 parameters:

- `a` - An array with the parameters of the function.
- `x` - The sample data to be fit.

Our main goal is to find the best suitable `a` that match given `x` values to given `y` values.

Parameters

- **fit_func** (*callable*) – The actual fitting function.
- **n** (*int*) – Number of parameters. the length of "a" in *fit_func*.
- **name** (*str*) – The name of the function.
- **syntax** (*str*) – The syntax of the fitting function
- **a_derivative** (*callable*) – a function representing the derivative of fit_func according to the "a" array
- **x_derivative** (*callable*) – a function representing the derivative of fit_func according to x
- **save** (*bool*) – Should this function be saved in the `FittingFunctionsRegistry`

active_parameters
Property of number of active parameters.

Returns number of active parameters (aka, unfixed).

Return type int

assign(*a: Union[List[float], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a81300d0>]*) → eddington.fitting_function_class.FittingFunction
Assign the function parameters.

Parameters **a** (*list of floats or np.ndarray*) – Parameters to be assigned

Returns self

Return type *FittingFunction*

clear_fixed() → eddington.fitting_function_class.FittingFunction
Clear all fixed parameters.

Returns self

Return type *FittingFunction*

fix(*index*, *value*)
Fix parameter with predefined value.

Parameters

- **index** (*int*) – The index of the parameter to fix. Starting from 0
- **value** (*float*) – The value to fix

Returns self *FittingFunction*

Raises **FittingFunctionRuntimeError** – Raised when trying to fix a non existing parameter.

title_name
Function name in title format.

Returns title name

Return type str

unfix(*index*)
Unfix a fixed parameter.

Parameters **index** (*int*) – The index of the parameter to unfix

Returns self *FittingFunction*

fitting_function decorator

fitting_function_class.**fitting_function**(*name: Optional[str] = None, syntax: Optional[str] = None, a_derivative: Optional[Callable[[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a81278d0>, Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8127910>, float]], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8127f50>]] = None, x_derivative: Optional[Callable[[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130290>, Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130210>, float]], Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130610>, float]]] = None, save: bool = True*) → Callable[[Callable[[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130910>, Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130950>, float]], Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8130a10>, float]]], eddington.fitting_function_class.FittingFunction]

Wrapper making a simple callable into a *FittingFunction*.

Parameters

- **n** (*int*) – Number of parameters. The length of parameter a of the fitting function.
- **name** (*str*) – The name of the function.
- **syntax** (*str*) – The syntax of the fitting function.
- **a_derivative** (*callable*) – a function representing the derivative of the fitting function according to the "a" parameter array
- **x_derivative** – a function representing the derivative of the fitting function according to x
- **save** (*bool*) – Should this function be saved in the FittingFunctionsRegistry

Returns a fitting function

Return type *FittingFunction*

1.3.3 Fitting Functions Registry

class eddington.fitting_functions_registry.**FittingFunctionsRegistry**

A singleton class containing all saved FittingFunction instances.

classmethod add(*func*)

Add a fitting function.

Parameters **func** (FittingFunction) – fitting function to add to registry

Raises **FittingFunctionSaveError** – Raised when trying to add a function with a name which already exists.

classmethod all()
Get all fitting functions.

Returns list of all fitting functions

Return type List[*FittingFunction*]

classmethod clear() → None
Clear all fitting functions.

classmethod exists(*func_name: str*) → bool
Checks whether a fitting function exist.

Parameters **func_name** (*str*) – Name of the function to load.

Returns bool

classmethod load(*func_name: str*)
Get a fitting function by name.

Parameters **func_name** (*str*) – Name of the function to load.

Returns a fitting function from the registry

Return type *FittingFunction*

Raises **FittingFunctionLoadError** – Raised when trying to load a function which does not exist in the registry

classmethod names() → List[str]
Property of the names of all fitting functions.

Returns Names of all fitting functions in registry

Return type List[str]

classmethod remove(*func_name: str*) → None
Remove a fitting function.

Parameters **func_name** (*str*) – Name of the function to remove.

1.3.4 Fitting Result

class eddington.fitting_result.**FittingResult**(*a0: Union[List[float], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8138fd0>], a: Union[List[float], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a81370d0>], aerr: Union[List[float], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8137190>], acov: Union[List[List[float]], <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a8137450>], degrees_of_freedom: int, chi2: float, precision: int = 3*)
Dataclass that contains the relevant parameters returned by a fitting algorithm.

Parameters

- **a0** (*list of floats or np.ndarray*) – The initial guess for the fitting function parameters.
- **a** (*list of floats or np.ndarray*) – The result for the fitting parameters.
- **aerr** (*list of floats or np.ndarray*) – Estimated errors of a.
- **arerr** (*list of floats or np.ndarray*) – Estimated relative errors of a (equivalent to aerr/a).
- **acov** (*list of lists of floats or np.ndarray*) – Covariance matrix of a.
- **degrees_of_freedom** (*int*) – How many degrees of freedom of the fittings.
- **chi2** (*float*) – Optimization evaluation for the fit.
- **chi2_reduced** (*float*) – Reduced chi2.
- **p_probability** (*float*) – P-probability (p-value) of the fitting, evaluated from chi2_reduced.

json_string
Json representation string.

Returns self representing json string

Return type str

precision = 3

pretty_string
Pretty representation string.

Returns self representing pretty string

Return type str

save_json(*file_path: Union[str, pathlib.Path]*) → None
Write the result to a json file.

Parameters **file_path** (str or Path) – Path to write the result in. if None, prints to console.

save_txt(*file_path: Union[str, pathlib.Path]*) → None
Write the result to a text file.

Parameters **file_path** (str or Path) – Path to write the result in. if None, prints to console.

1.3.5 Fit To Data

fitting.**fit**(*func: eddington.fitting_function_class.FittingFunction*, *a0: <sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a815c710> = None*, *use_x_derivative: bool = True*, *use_a_derivative: bool = True*) → eddington.fitting_result.FittingResult
Implementation of the fitting algorithm.

This functions wraps *scipy*'s ODR algorithm.

Parameters

- **data** (FittingData) – Fitting data to optimize
- **func** (FittingFunction) – a function to fit the data according to.
- **a0** (*np.ndarray*) – initial guess for the parameters
- **use_x_derivative** (*bool*) – indicates whether to use x derivative or not.

- **use_a_derivative** (*bool*) – indicates whether to use a derivative or not.

Returns FittingResult

Raises **FittingError** – Raised when missing information for the fitting algorithm.

1.3.6 Out-of-the-Box Fitting Functions

fitting_functions_list.**linear**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2050>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2110>, float]

Simple linear fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] + a[1] * x`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 2
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

fitting_functions_list.**constant**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2350>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2410>, float]

Constant fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 1
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

fitting_functions_list.**parabolic**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2650>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2710>, float]

Parabolic fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] + a[1] * x + a[2] * x ^ 2`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 3
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

fitting_functions_list.**straight_power**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2950>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2a10>, float]

Represent fitting of y ~ x^n.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * (x + a[1]) ^ a[2] + a[3]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 4
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

fitting_functions_list.**inverse_power**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2c50>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2d10>, float]

Represent fitting of y ~ x^(-n).

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] / (x + a[1]) ^ a[2] + a[3]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 4
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

`fitting_functions_list.`**`polynomial`**`()` → eddington.fitting_function_class.FittingFunction

Creates a polynomial fitting function with parameters as coefficients.

Parameters **n** (*int*) – Degree of the polynomial.

Returns a polynomial fitting function

Return type *FittingFunction*

Raises **FittingFunctionLoadError** – Raised when trying to load a polynomial with negative degree.

`fitting_functions_list.`**`hyperbolic`**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c2f50>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8050>, float]

Hyperbolic fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] / (x + a[1]) + a[2]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 3
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

`fitting_functions_list.`**`exponential`**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8290>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8350>, float]

Exponential fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * exp(a[1] * x) + a[2]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 3
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

`fitting_functions_list.`**`sin`**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8890>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8950>, float]

Sine fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * sin(a[1] * x + a[2]) + a[3]`

Parameters

- **a** (`numpy.ndarray`) – Coefficients array of length 4
- **x** (`numpy.ndarray` or `float`) – Free parameter

Returns `numpy.ndarray` or `float`

`fitting_functions_list.`**`cos`**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8590>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8650>, float]

Cosines fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * cos(a[1] * x + a[2]) + a[3]`

Parameters

- **a** (numpy.ndarray) – Coefficients array of length 4
- **x** (numpy.ndarray or float) – Free parameter

Returns numpy.ndarray or float

fitting_functions_list.**normal**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8b90>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8c50>, float]

Normal distribution fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * exp( - ((x - a[1]) / a[2]) ^ 2) + a[3]`

Parameters

- **a** (numpy.ndarray) – Coefficients array of length 4
- **x** (numpy.ndarray or float) – Free parameter

Returns numpy.ndarray or float

fitting_functions_list.**poisson**(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8e90>, float]*) → Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80c8f50>, float]

Poisson fitting function.

param a Parameters to be fitted

type a np.ndarray

param x Value to be evaluated by the function

type x float or np.ndarray

return evaluation value or values

rtype float or np.ndarray

Syntax: `y = a[0] * (a[1] ^ x) * exp(-a[1]) / gamma(x+1) + a[2]`

Parameters

- **a** (numpy.ndarray) – Coefficients array of length 3
- **x** (numpy.ndarray or float) – Free parameter

Returns numpy.ndarray or float

1.3.7 Figure Builder

class eddington.plot.figure_builder.**FigureBuilder**(*title: dataclasses.InitVar = None*, *xlabel: dataclasses.InitVar = None*, *ylabel: dataclasses.InitVar = None*, *grid: dataclasses.InitVar = False*, *legend: dataclasses.InitVar = False*)

Builder class for creating a figure.

add_data(*data: eddington.fitting_data.FittingData*, *label: Optional[str] = None*, *color: Optional[str] = None*)

Add scatter to figure.

Parameters

- **data** – Data to plot
- **label** (*str*) – Label of the error bar to add to the legend.
- **color** (*str*) – Color of the error bar.

Returns self

Return type *FigureBuilder*

Raises **PlottingError** – Raised when data doesn't have x or y values

add_error_bar(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e8050>, List[float]]*, *xerr: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e8110>, List[float], None]*, *y: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e81d0>, List[float]]*, *yerr: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e8290>, List[float], None]*, *label: Optional[str] = None*, *color: Optional[str] = None*)

Add error bar to figure.

Parameters

- **x** (*floats list or numpy.ndarray*) – X values of the error bar
- **xerr** (*floats list or numpy.ndarray*) – X error values of the error bar
- **y** (*floats list or numpy.ndarray*) – Y values of the error bar
- **yerr** (*floats list or numpy.ndarray*) – Y error values of the error bar
- **label** (*str*) – Label of the error bar to add to the legend.
- **color** (*str*) – Color of the error bar.

Returns self

Return type *FigureBuilder*

add_grid()

Set grid lines to figure.

Returns self

Return type *FigureBuilder*

add_horizontal_line(*interval: eddington.interval.Interval*, *y_value: float*, *linestyle: eddington.plot.line_style.LineStyle = <LineStyle.SOLID: 'solid'>*, *color: Optional[str] = None*)

Add horizontal line to figure.

Parameters

- **interval** (*Interval*) – Interval representing x values
- **y_value** (*float*) – y value of the horizontal line
- **linestyle** (*LineStyle*) – Line style of the horizontal line
- **color** (*str*) – Optional. Color of the line style

Returns self

Return type *FigureBuilder*

add_instruction(*instruction: eddington.plot.figure_builder.FigureInstruction*) → eddington.plot.figure_builder.FigureBuilder

Add general instruction to plot building.

Parameters **instruction** (*FigureInstruction*) – Instruction instance for what to add to the figure

Returns self

Return type *FigureBuilder*

Raises **PlottingError** – Raised when trying to add a unique instruction for the second time.

add_legend()

Set legend lines to figure.

Returns self

Return type *FigureBuilder*

add_plot(*interval: eddington.interval.Interval, a: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e84d0>, List[float]], func: eddington.fitting_function_class.FittingFunction, label: Optional[str] = None, color: Optional[str] = None, linestyle: eddington.plot.line_style.LineStyle = <LineStyle.SOLID: 'solid'>*)

Add plot to figure.

Parameters

- **interval** (*Interval*) – Interval representing x values
- **func** (FittingFunction) – Function to show its plot
- **a** (*floats list or numpy.ndarray*) – parameters to use as input to fitting function
- **label** (*str*) – Label of the plot to add to the legend.
- **color** (*str*) – Optional. Color of the line style
- **linestyle** (*LineStyle*) – Line style of the horizontal line

Returns self

Return type *FigureBuilder*

add_scatter(*x: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e8350>, List[float]], y: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fb3a80e8410>, List[float]], label: Optional[str] = None, color: Optional[str] = None*)

Add scatter to figure.

Parameters

- **x** (*floats list or numpy.ndarray*) – X values of the error bar
- **y** (*floats list or numpy.ndarray*) – Y values of the error bar
- **label** (*str*) – Label of the error bar to add to the legend.
- **color** (*str*) – Color of the error bar.

Returns self

Return type *FigureBuilder*

add_title(*title: str*)
Add title instruction to figure.

Parameters **title** (*str*) – Title to add to the figure

Returns self

Return type *FigureBuilder*

add_x_log_scale()
Set x axis scale to logarithmic scale.

Returns self

Return type *FigureBuilder*

add_xlabel(*xlabel: str*)
Add x label instruction to figure.

Parameters **xlabel** (*str*) – X axis label to add to the figure

Returns self

Return type *FigureBuilder*

add_y_log_scale()
Set y axis scale to logarithmic scale.

Returns self

Return type *FigureBuilder*

add_ylabel(*ylabel: str*)
Add y label instruction to figure.

Parameters **ylabel** (*str*) – Y axis label to add to the figure

Returns self

Return type *FigureBuilder*

build() → eddington.plot.figure.Figure
Build figure.

Returns Built figure item

Return type Figure

grid = False

legend = False

title = None

xlabel = None

ylabel = None

1.4 Tutorials

1.4.1 Writing Data Files

Data File Formats

Eddington excepts data files in 3 formats: CSV, Excel, and Json. In this tutorial we will walk through the syntax of each format.

CSV File

CSV data files should have a headers row followed by records rows, each column represent a specific value of the record. For example:

```
a,delta a,b,delta b,c,delta c
10,0.5,16,1,100,14
20,1,29,1.3,401,10
30,1.2,47,0.8,910,11
40,0.3,56,2,1559,8
50,0.4,70,1.1,2480,10
60,1.1,92,0.2,3623,5
70,1.3,100,2,4910,16
```

Excel File

Those instructions are the same for Excel files:

	A	B	C	D	E	F
1	a	delta a	b	delta b	c	delta c
2	10	0.5	16	1	100	14
3	20	1	29	1.3	401	10
4	30	1.2	47	0.8	910	11
5	40	0.3	56	2	1559	8
6	50	0.4	70	1.1	2480	10
7	60	1.1	92	0.2	3623	5
8	70	1.3	100	2	4910	16

Json File

As for Json files, it should be written as a dictionary, with each header as the key mapped to it's values. For example:

```
{
  "a": [10, 20, 30, 40, 50, 60, 70],
  "delta_a": [0.5, 1.0, 1.2, 0.3, 0.4, 1.1, 1.3],
  "b": [16.0, 29.0, 47.0, 56.0, 70.0, 92.0, 100.0],
  "delta_b": [1.0, 1.3, 0.8, 2.0, 1.1, 0.2, 2.0],
  "c": [100.0, 401.0, 910.0, 1559.0, 2480.0, 3623.0, 4910.0],
```

(continues on next page)

(continued from previous page)

```
  "delta_d": [14.0, 10.0, 11.0, 8.0, 10.0, 5.0, 16.0]
}
```

Warning: The order of each values list should be the orders of the records. Mismatching the order may cause fitting errors.

Specify The Data Columns

Whatever the data file format you choose, you should specify the columns to be used for the fitting. By default the first 4 columns are used as the *x*, *x error*, *y* and *y error* in that order. That means that in our example "a" will be taken as *x*, "delta_a" will be taken as the *x error*, "b" will be taken as *y* and "delta_b" will be taken as the *y error*.

Common Errors

If you encounter a loading error while trying to load a data file, it may be caused by one of the following problems:

- One or more of your records has non-float value or it is blank.
- In Excel files, you may have added an additional cell outside of the records table.
- In Excel and CSV files, you might have missed adding a headers row
- In Json, one or more of your columns has different length than the others (which means a record is missing a value).

1.4.2 Writing Your Own Fitting Function

When Should You Do It?

Even though Eddington offers many default fitting functions, sometimes you may want to customize your own fitting function.

Consider the following case: You conduct an experiment to demonstrate the *Thin Lens Equation*:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

After rearranging the equation you get:

$$v = \frac{uf}{u - f}$$

You have data records of v and u and you want to estimate f. You **can** use our *hyperbolic* fitting function, but in order to find f You'll have to do some more calculations with respect to the errors of the parameters you've found.

A somewhat easier approach would be to use the following fitting function:

$$y = \frac{a_0 x}{x - a_0} + a_1$$

Now, after you fit the data, you get f directly (which equals to a_0)

Since this fitting function is not implemented by default, you'd have to implement it yourself.

Basic implementation

A basic implementation of the fitting function presented in the example above would look like this:

```
from eddington import fitting_function

@fitting_function(n=2)
def lens(a, x):
    return (a[0] * x) / (x - a[0]) + a[1]
```

We wrap the `lens` fitting function with the `fitting_function` decorator in order to indicate that this function is actually a fitting function. the `n` variable indicates how many parameters the fitting function expects. In our example, we expect 2 parameters: `a[0]` which is f, and `a[1]` which encapsulates the systematic errors in our v samples.

Note: The inputs of the fitting function are `a` which is the parameters vector and `x` which is the free variable. While `a` can be from any array-like type such as `list`, `tuple`, `numpy.ndarray`, etc. `x` can be both a `numpy.ndarray` and `float`.

Now, we can use the fitting function we've created in order to fit the data:

```
from eddington import FittingData, fit

fitting_data = FittingData.read_from_csv("/path/to/data.csv")  # Load data from file.
fitting_result = fit(fitting_data, lens)  # Do the actual fitting
print(fitting_result)  # Print the results
```

This usage is more than enough for most use-cases.

Derivatives

Sometimes, you wish to get an accurate fit, and **fast**. One way to achieve that is to add derivatives to the fitting function. In our example, we have the following derivatives:

x *derivative* -

$$\frac{\partial y}{\partial x} = -\frac{a_0^2}{(x-a_0)^2}$$

a_0 *derivative* -

$$\frac{\partial y}{\partial a_0} = \frac{x^2}{(x-a_0)^2}$$

a_1 *derivative* -

$$\frac{\partial y}{\partial a_1} = 1$$

In order to add those derivatives to the fitting function, we should add the `x_derivative` and `a_derivative` to the `fitting_function` decorator. In our example:

```
import numpy as np
from eddington import fitting_function, FittingData, fit


@fitting_function(
```

(continues on next page)

(continued from previous page)

```
    n=2,
    x_derivative=lambda a, x: -np.power(a[0], 2) / np.power(x - a[0], 2),
    a_derivative=lambda a, x: np.stack(
        [
            np.power(x, 2) / np.power(x - a[0], 2),
            np.ones(shape=np.shape(x)),
        ]
    ),
)
def lens(a, x):
    return (a[0] * x) / (x - a[0]) + a[1]
```

Note: When implementing the derivatives pay attention that you take `a` as the first parameter and `x` as the second. Moreover, make sure that the *dimension* of the output `x_derivative` returns a `numpy.ndarray` with dimension similar to `x`, while `a_derivative` returns a `numpy.ndarray` with dimension equal to `x` dimension times `a` dimension.

The Fitting Functions Registry

By default, creating a new fitting function adds it automatically to the *FittingFunctionsRegistry*, a singleton containing all fitting functions. Once the fitting function you've created is imported (for example, in the *__init__.py* file) it can be loaded from the registry in the following way:

```
from eddington import FittingFunctionsRegistry

fit_func = FittingFunctionsRegistry.load("lens")
```

If you wish to specify a different name to the fitting function by which it can be loaded from the registry, use the *name* parameter in the *fitting_function* decorator in the following way:

```
from eddington import fitting_function, FittingFunctionsRegistry

@fitting_function(n=2, name="my_amazing_func")
def lens(a, x):
    return (a[0] * x) / (x - a[0]) + a[1]

fit_func = FittingFunctionsRegistry.load("my_amazing_func")  # Returns the "lens"␣
↪function
```

If you expect others to use your new fitting function, consider adding a *syntax* string indicating how the fitting functions fit the data. This can be printed out when needed. For example:

```
from eddington import fitting_function, FittingFunctionsRegistry

@fitting_function(n=2, syntax="(a[0] * x) / (x - a[0]) + a[1]")
def lens(a, x):
    return (a[0] * x) / (x - a[0]) + a[1]

...

fit_func = FittingFunctionsRegistry.load("lens")
print(f"Syntax is: {fit_func.syntax}")  # Prints out the defined syntax
```

Lastly, if you wish the fitting function to not be saved into the registry, specify `save=False` in the *fitting_function* decorator. For example:

```
from eddington import fitting_function

@fitting_function(n=2, save=False)
def lens(a, x):
    return (a[0] * x) / (x - a[0]) + a[1]
```

As mentioned earlier, by default *save* is set to `True`.

Warning: Two functions cannot be saved into the registry under the same name. Make sure that every new fitting function you write has a unique name, which is not one of the default fitting functions or another custom fitting function you expect to use in your code.

Using External Packages

When writing a custom fitting function, one can use external packages such as `numpy` and `scipy` in the fitting code. Here is an example:

```
import numpy as np
from eddington import fitting_function

@fitting_function(n=2)
def test(a, x):
    return a[0] / np.sqrt(x - a[0])
```

In that way you can make more complex fitting functions.

Note: The optimization algorithm uses `numpy.array` in order to make parallel calculations. Therefore, using the built-in `math` package will fail most of the times. In order to solve the problem, use `numpy` methods (which *do* excepts arrays) instead.

Index

A

active_parameters (*eddington.fitting_function_class.FittingFunction attribute*), 17
add() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 19
add_data() (*eddington.plot.figure_builder.FigureBuilder method*), 27
add_error_bar() (*eddington.plot.figure_builder.FigureBuilder method*), 27
add_grid() (*eddington.plot.figure_builder.FigureBuilder method*), 27
add_horizontal_line() (*eddington.plot.figure_builder.FigureBuilder method*), 27
add_instruction() (*eddington.plot.figure_builder.FigureBuilder method*), 28
add_legend() (*eddington.plot.figure_builder.FigureBuilder method*), 28
add_plot() (*eddington.plot.figure_builder.FigureBuilder method*), 28
add_scatter() (*eddington.plot.figure_builder.FigureBuilder method*), 28
add_title() (*eddington.plot.figure_builder.FigureBuilder method*), 29
add_x_log_scale() (*eddington.plot.figure_builder.FigureBuilder method*), 29
add_xlabel() (*eddington.plot.figure_builder.FigureBuilder method*), 29
add_y_log_scale() (*eddington.plot.figure_builder.FigureBuilder method*), 29
add_ylabel() (*eddington.plot.figure_builder.FigureBuilder method*), 29
all() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 19
all_columns (*eddington.fitting_data.FittingData attribute*), 10
all_records (*eddington.fitting_data.FittingData attribute*), 10
all_selected() (*eddington.fitting_data.FittingData method*), 10
assign() (*eddington.fitting_function_class.FittingFunction method*), 17

B

build() (*eddington.plot.figure_builder.FigureBuilder method*), 29

C

cell_data() (*eddington.fitting_data.FittingData method*), 10
clear() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 20
clear_fixed() (*eddington.fitting_function_class.FittingFunction method*), 18
column_data() (*eddington.fitting_data.FittingData method*), 10
column_domain() (*eddington.fitting_data.FittingData method*), 10
constant() (*eddington.fitting_functions_list method*), 22
copy() (*eddington.fitting_data.FittingData method*), 11
cos() (*eddington.fitting_functions_list method*), 25

D

data (*eddington.fitting_data.FittingData attribute*), 11

E

exists() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 20
exponential() (*eddington.fitting_functions_list method*), 24

F

FigureBuilder (*class in eddington.plot.figure_builder*), 27
fit() (*eddington.fitting method*), 21
fitting_function() (*eddington.fitting_function_class method*), 19
FittingData (*class in eddington.fitting_data*), 9
FittingFunction (*class in eddington.fitting_function_class*), 17
FittingFunctionsRegistry (*class in eddington.fitting_functions_registry*), 19
FittingResult (*class in eddington.fitting_result*), 20
fix() (*eddington.fitting_function_class.FittingFunction method*), 18

G

grid (*eddington.plot.figure_builder.FigureBuilder attribute*), 29

H

hyperbolic() (*eddington.fitting_functions_list method*), 24

I

inverse_power() (*eddington.fitting_functions_list method*), 23
is_selected() (*eddington.fitting_data.FittingData method*), 11

J

json_string (*eddington.fitting_result.FittingResult attribute*), 21

L

legend (*eddington.plot.figure_builder.FigureBuilder attribute*), 29
linear() (*eddington.fitting_functions_list method*), 22
load() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 20

N

names() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 20
non_selected() (*eddington.fitting_data.FittingData method*), 11
normal() (*eddington.fitting_functions_list method*), 26
number_of_columns (*eddington.fitting_data.FittingData attribute*), 11
number_of_records (*eddington.fitting_data.FittingData attribute*), 11

P

parabolic() (*eddington.fitting_functions_list method*), 22
poisson() (*eddington.fitting_functions_list method*), 26
polynomial() (*eddington.fitting_functions_list method*), 24
precision (*eddington.fitting_result.FittingResult attribute*), 21
pretty_string (*eddington.fitting_result.FittingResult attribute*), 21

R

read_from_csv() (*eddington.fitting_data.FittingData class method*), 11
read_from_excel() (*eddington.fitting_data.FittingData class method*), 12
read_from_json() (*eddington.fitting_data.FittingData class method*), 12
record_data() (*eddington.fitting_data.FittingData method*), 13
records (*eddington.fitting_data.FittingData attribute*), 13
records_indices (*eddington.fitting_data.FittingData attribute*), 13
remove() (*eddington.fitting_functions_registry.FittingFunctionsRegistry class method*), 20
residuals() (*eddington.fitting_data.FittingData method*), 13

S

save_csv() (*eddington.fitting_data.FittingData method*), 13
save_excel() (*eddington.fitting_data.FittingData method*), 13
save_json() (*eddington.fitting_result.FittingResult method*), 21
save_txt() (*eddington.fitting_result.FittingResult method*), 21
select_all_records() (*eddington.fitting_data.FittingData method*), 13
select_by_domains() (*eddington.fitting_data.FittingData method*), 13
select_by_x_domain() (*eddington.fitting_data.FittingData method*), 14

select_by_y_domain() (*eddington.fitting_data.FittingData method*), 14
select_record() (*eddington.fitting_data.FittingData method*), 14
set_cell() (*eddington.fitting_data.FittingData method*), 14
set_header() (*eddington.fitting_data.FittingData method*), 15
sin() (*eddington.fitting_functions_list method*), 25
statistics() (*eddington.fitting_data.FittingData method*), 15
statistics_map (*eddington.fitting_data.FittingData attribute*), 15
straight_power() (*eddington.fitting_functions_list method*), 23

T

title (*eddington.plot.figure_builder.FigureBuilder attribute*), 29
title_name (*eddington.fitting_function_class.FittingFunction attribute*), 18

U

unfix() (*eddington.fitting_function_class.FittingFunction method*), 18
unselect_all_records() (*eddington.fitting_data.FittingData method*), 15
unselect_record() (*eddington.fitting_data.FittingData method*), 15
used_columns (*eddington.fitting_data.FittingData attribute*), 15

X

x (*eddington.fitting_data.FittingData attribute*), 15
x_column (*eddington.fitting_data.FittingData attribute*), 15
x_domain (*eddington.fitting_data.FittingData attribute*), 15
x_index (*eddington.fitting_data.FittingData attribute*), 16
xerr (*eddington.fitting_data.FittingData attribute*), 16
xerr_column (*eddington.fitting_data.FittingData attribute*), 16
xerr_index (*eddington.fitting_data.FittingData attribute*), 16
xlabel (*eddington.plot.figure_builder.FigureBuilder attribute*), 29

Y

y (*eddington.fitting_data.FittingData attribute*), 16
y_column (*eddington.fitting_data.FittingData attribute*), 16
y_domain (*eddington.fitting_data.FittingData attribute*), 16
y_index (*eddington.fitting_data.FittingData attribute*), 16
yerr (*eddington.fitting_data.FittingData attribute*), 16
yerr_column (*eddington.fitting_data.FittingData attribute*), 16
yerr_index (*eddington.fitting_data.FittingData attribute*), 17
ylabel (*eddington.plot.figure_builder.FigureBuilder attribute*), 29