

---

# **MadMiner Documentation**

***Release 0.9.5***

**Johann Brehmer, Felix Kling, Irina Espejo, Sinclert Perez, Kyle Cr**

**Dec 19, 2022**



## SITES

<b>1</b>	<b>Introduction to MadMiner</b>	<b>3</b>
<b>2</b>	<b>Getting started</b>	<b>5</b>
<b>3</b>	<b>Using MadMiner</b>	<b>7</b>
<b>4</b>	<b>Trouble-shooting</b>	<b>11</b>
<b>5</b>	<b>References</b>	<b>13</b>
<b>6</b>	<b>madminer.analysis package</b>	<b>15</b>
<b>7</b>	<b>madminer.core package</b>	<b>19</b>
<b>8</b>	<b>madminer.delphes package</b>	<b>29</b>
<b>9</b>	<b>madminer.fisherinformation package</b>	<b>37</b>
<b>10</b>	<b>madminer.lhe package</b>	<b>55</b>
<b>11</b>	<b>madminer.likelihood package</b>	<b>63</b>
<b>12</b>	<b>madminer.limits package</b>	<b>71</b>
<b>13</b>	<b>madminer.ml package</b>	<b>81</b>
<b>14</b>	<b>madminer.plotting package</b>	<b>113</b>
<b>15</b>	<b>madminer.sampling package</b>	<b>123</b>
<b>16</b>	<b>Indices and tables</b>	<b>139</b>
	<b>Python Module Index</b>	<b>141</b>
	<b>Index</b>	<b>143</b>



*Johann Brehmer, Felix Kling, Irina Espejo, and Kyle Cranmer*

**Machine learning–based inference for particle physics**



## **INTRODUCTION TO MADMINER**

Particle physics processes are usually modelled with complex Monte-Carlo simulations of the hard process, parton shower, and detector interactions. These simulators typically do not admit a tractable likelihood function: given a (potentially high-dimensional) set of observables, it is usually not possible to calculate the probability of these observables for some model parameters. Particle physicists usually tackle this problem of “likelihood-free inference” by hand-picking a few “good” observables or summary statistics and filling histograms of them. But this conventional approach discards the information in all other observables and often does not scale well to high-dimensional problems.

In the three publications “[Constraining Effective Field Theories with Machine Learning](#)”, “[A Guide to Constraining Effective Field Theories with Machine Learning](#)”, and “[Mining gold from implicit models to improve likelihood-free inference](#)”, a new approach has been developed. In a nutshell, additional information is extracted from the simulations that is closely related to the matrix elements that determine the hard process. This “augmented data” can be used to train neural networks to efficiently approximate arbitrary likelihood ratios. We playfully call this process “mining gold” from the simulator, since this information may be hard to get, but turns out to be very valuable for inference.

But the gold does not have to be hard to mine. This package automates these inference strategies. It wraps around the simulators MadGraph and Pythia, with different options for the detector simulation. All steps in the analysis chain from the simulation to the extraction of the augmented data, their processing, and the training and evaluation of the neural estimators are implemented.





## GETTING STARTED

### 2.1 Simulator dependencies

Make sure the following tools are installed and running:

- MadGraph (we have tested our setup with version 2.8.0+). See [MadGraph's website](#) for installation instructions. Note that MadGraph requires a Fortran compiler as well as Python 3.7+.
- For the analysis of systematic uncertainties, LHAPDF6 has to be installed with Python support (see also [the documentation of MadGraph's systematics tool](#)).

For the detector simulation part, there are different options. For simple parton-level analyses, we provide a bare-bones option to calculate truth-level observables which do not require any additional packages. We have also implemented a fast detector simulation based on Delphes with a flexible framework to calculate observables. Using this adds additional requirements:

```
echo "install pythia8" | python3 <MadGraph_dir>/bin/mg5_aMC  
echo "install Delphes" | python3 <MadGraph_dir>/bin/mg5_aMC
```

Finally, Delphes can be replaced with another detector simulation, for instance a full detector simulation based with Geant4. In this case, the user has to implement code that runs the detector simulation, calculates the observables, and stores the observables and weights in the HDF5 file. The DelphesProcessor and LHEProcessor classes might provide some guidance for this.

### 2.2 Install MadMiner

To install the MadMiner package with all its Python dependencies, run `pip install madminer`.

To get the latest development version as well as the tutorials, clone the [GitHub repository](#) and run `pip install -e .` from the repository main folder.



## USING MADMINER

We provide different resources that help with the use of MadMiner:

### 3.1 Paper

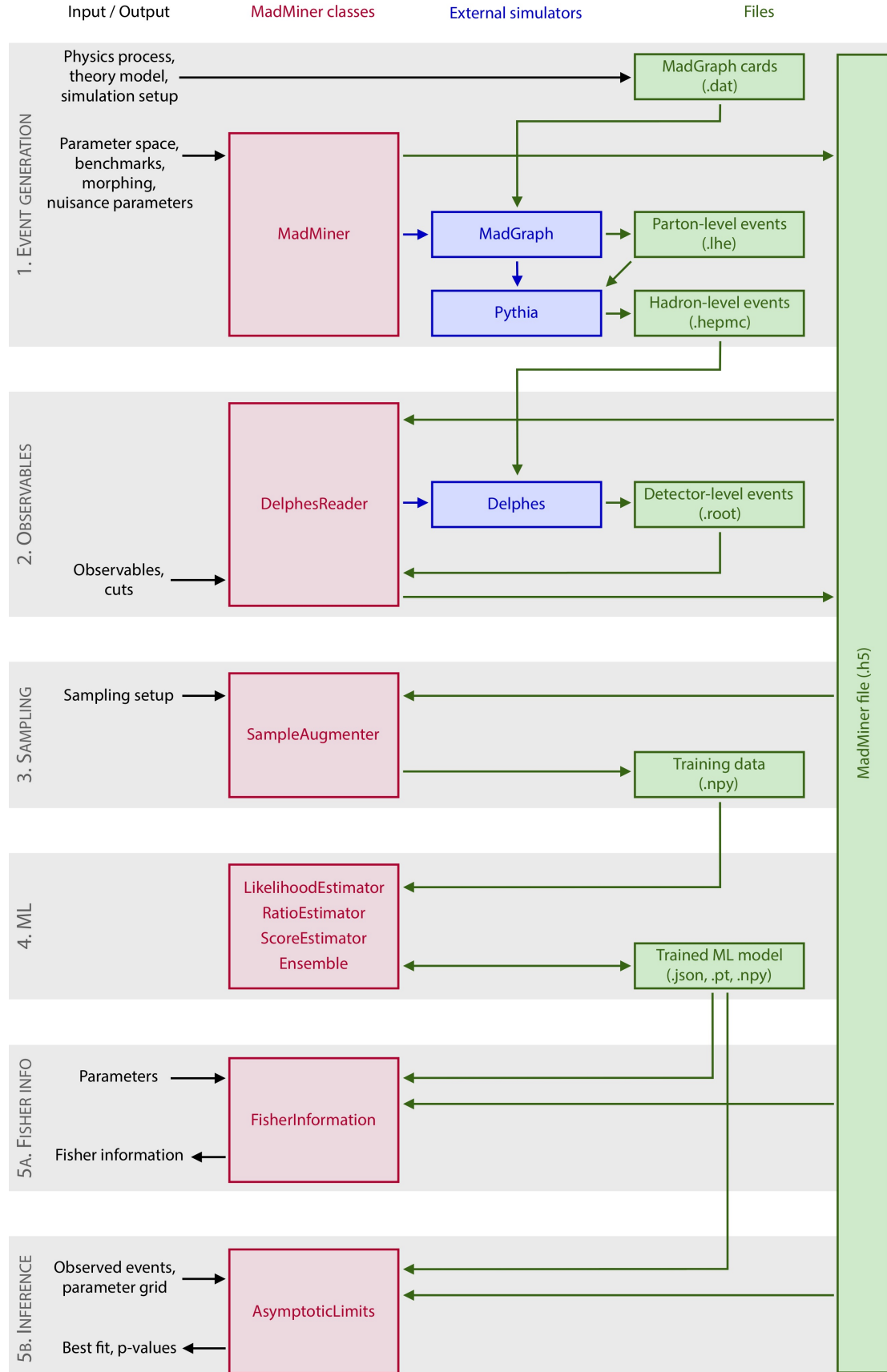
Our main publication [MadMiner: Machine-learning-based inference for particle physics](#) provides an overview over this package. We recommend reading it first before jumping into the code.

### 3.2 Tutorials

In the [examples](#) folder in this repository, we provide two tutorials. The first is called *Toy simulator*, and it is based on a toy problem rather than a full particle-physics simulation. It demonstrates inference with MadMiner without spending much time on the more technical steps of running the simulation. The second, called *Particle physics*, shows all steps of a particle-physics analysis with MadMiner.

### 3.3 Typical workflow

Here we illustrate the structure of data analysis with MadMiner:



- `madminer.core` contains the functions to set up the process, parameter space, morphing, and to steer MadGraph and Pythia.
- `madminer.lhe` and `madminer.delphes` contain two example implementations of a detector simulation and observable calculation. This part can easily be swapped out depending on the use case.
- In `madminer.sampling`, train and test samples for the machine learning part are generated and augmented with the joint score and joint ratio.
- `madminer.ml` contains an implementation of the machine learning part. The user can train and evaluate estimators for the likelihood ratio or score.
- Finally, `madminer.fisherinformation` contains functions to calculate the Fisher information, both on parton level or detector level, in the full process, individual observables, or the total cross section.

## 3.4 Technical documentation

The madminer API is documented on here as well, just look through the pages linked on the left.

## 3.5 Support

If you have any questions, please chat to us in our [Gitter community](#).



## TROUBLE-SHOOTING

If you are having issues with MadMiner, please go through the following check list:

### 4.1 Event generation crashing

- Is MadGraph correctly installed? Can you generate events with MadGraph on its own, including the reweighing option?
- If you are using Pythia and Delphes: Are their installations working? Can you run MadGraph with Pythia, and can you run Delphes on the resulting HepMC sample?
- If you are using PDF or scale uncertainties: Is LHAPDF installed with Python support?

### 4.2 Key errors when reading LHE files

- Do LHE files contain multiple weights, one for each benchmark, for each event?

### 4.3 Zero events after reading LHE or Delphes file

- Are there typos in the definitions of required observables, cuts, or efficiencies? If an observable, cut, or efficiency causes all events to be discarded, DEBUG-level logging output should help you narrow down the source.

### 4.4 Neural network output does not make sense

- Start simple: one or two hidden layers are often enough for a start.
- Does the loss go down during training? If not, try changing the learning rate.
- Are the loss on the training and validation sample very different? This is the trademark sign of over-training. Try a simpler network architecture, more data, or early stopping.





## REFERENCES

### 5.1 Citations

If you use MadMiner, please cite our main publication,

```
@article{Brehmer:2019xox,  
  author      = "Brehmer, Johann and Kling, Felix and Espejo, Irina and Cranmer, ↵  
↵ Kyle",  
  title       = "{MadMiner: Machine learning-based inference for particle physics}  
↵",  
  journal     = "Comput. Softw. Big Sci.",  
  volume      = "4",  
  year        = "2020",  
  number      = "1",  
  pages       = "3",  
  doi         = "10.1007/s41781-020-0035-2",  
  eprint      = "1907.10621",  
  archivePrefix = "arXiv",  
  primaryClass = "hep-ph",  
  SLACcitation = "%%CITATION = ARXIV:1907.10621;%%"  
}
```

The code itself can be cited as

```
@misc{MadMiner_code,  
  author      = "Brehmer, Johann and Kling, Felix and Espejo, Irina and Cranmer, ↵  
↵ Kyle",  
  title       = "{MadMiner}",  
  doi         = "10.5281/zenodo.1489147",  
  url         = {https://github.com/madminer-tool/madminer}  
}
```

The main references for the implemented inference techniques are the following:

- CARL: 1506.02169.
- MAF: 1705.07057.
- CASCAL, RASCAL, ROLR, SALLY, SALLINO, SCANDAL:
  - 1805.00013.
  - 1805.00020.

- 1805.12244.
- ALICE, ALICES: 1808.00973.

## 5.2 Acknowledgements

We are immensely grateful to all [contributors](#) and bug reporters! In particular, we would like to thank Zubair Bhatti, Philipp Englert, Lukas Heinrich, Alexander Held, Samuel Homiller and Duccio Pappadopulo.

The SCANDAL inference method is based on [Masked Autoregressive Flows](#), where our implementation is a PyTorch port of the original code by George Papamakarios, available at [this repository](#).

## MADMINER.ANALYSIS PACKAGE

### 6.1 Submodules

### 6.2 madminer.analysis.dataanalyzer module

**class** `madminer.analysis.dataanalyzer.DataAnalyzer`(*filename*, *disable\_morphing=False*,  
*include\_nuisance\_parameters=True*)

Bases: object

Collects common functionality that is used when analysing data in the MadMiner file.

#### Parameters

##### **filename**

[str] Path to MadMiner file (for instance the output of *madminer.delphes.DelphesProcessor.save()*).

##### **disable\_morphing**

[bool, optional] If True, the morphing setup is not loaded from the file. Default value: False.

##### **include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

#### Methods

<code>event_loader</code> ([start, end, batch_size, ...])	Yields batches of events in the MadMiner file.
<code>weighted_events</code> ([theta, nu, start_event, ...])	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients</code> (thetas[, nus, partition, ...])	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs</code> ([thetas, nus, partition, test_split, ...])	Returns the total cross sections for benchmarks or parameter points.

**event\_loader**(*start=0*, *end=None*, *batch\_size=100000*, *include\_nuisance\_parameters=None*,  
*generated\_close\_to=None*, *return\_sampling\_ids=False*)

Yields batches of events in the MadMiner file.

#### Parameters

##### **start**

[int, optional] First event index to load

**end**

[int or None, optional] Last event index to load

**batch\_size**

[int, optional] Batch size

**include\_nuisance\_parameters**

[bool, optional] Whether nuisance parameter benchmarks are included in the returned data

**generated\_close\_to**

[None or ndarray, optional] If None, this function yields all events. Otherwise, it just yields just the events that were generated at the closest benchmark point to a given parameter point.

**return\_sampling\_ids**

[bool, optional] If True, the iterator returns the sampling IDs in addition to observables and weights.

**Yields**

**observations**

[ndarray] Event data

**weights**

[ndarray] Event weights

**sampling\_ids**

[int] Sampling IDs (benchmark used for sampling for signal events, -1 for background events). Only returned if return\_sampling\_ids = True was set.

**weighted\_events**(*theta=None, nu=None, start\_event=None, end\_event=None, derivative=False, generated\_close\_to=None, n\_draws=None*)

Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.

**Parameters**

**theta**

[None or ndarray or str, optional] If None, the function returns all benchmark weights. If str, the function returns the weights for a given benchmark name. If ndarray, it uses morphing to calculate the weights for this value of theta. Default value: None.

**nu**

[None or ndarray, optional] If None, the nuisance parameters are set to their nominal values. Otherwise, and if theta is an ndarray, sets the values of the nuisance parameters.

**start\_event**

[int] Index (in the MadMiner file) of the first event to consider.

**end\_event**

[int] Index (in the MadMiner file) of the last unweighted event to consider.

**derivative**

[bool, optional] If True and if theta is not None, the derivative of the weights with respect to theta are returned. Default value: False.

**generated\_close\_to**

[None or int, optional] Only returns benchmarks generated from this benchmark (and background events). Default value: None.

**n\_draws**

[None or int, optional] If not None, returns only this number of events, drawn randomly.

**Returns**

**x**[ndarray] Observables with shape  $(n\_unweighted\_samples, n\_observables)$ .**weights**[ndarray] If theta is None and derivative is False, benchmark weights with shape  $(n\_unweighted\_samples, n\_benchmarks)$  in pb. If theta is not None and derivative is True, the gradient of the weight for the given parameter with respect to theta with shape  $(n\_unweighted\_samples, n\_gradients)$  in pb. Otherwise, weights for the given parameter theta with shape  $(n\_unweighted\_samples,)$  in pb.**xsec\_gradients**(*thetas, nus=None, partition='all', test\_split=0.2, validation\_split=0.2, gradients='all', batch\_size=100000, generated\_close\_to=None*)

Returns the gradient of total cross sections with respect to parameters.

**Parameters****thetas**

[list of (ndarray or str), optional] If None, the function returns all benchmark cross sections. Otherwise, it returns the cross sections for a series of parameter points that are either given by their benchmark name (as a str), their benchmark index (as an int), or their parameter value (as an ndarray, using morphing). Default value: None.

**nus**

[None or list of (None or ndarray), optional] If None, the nuisance parameters are set to their nominal values (0), i.e. no systematics are taken into account. Otherwise, the list has to have the same number of elements as thetas, and each entry can specify nuisance parameters at nominal value (None) or a value of the nuisance parameters (ndarray).

**partition**

[{"train", "test", "validation", "all"}, optional] Which events to use. Default: "all".

**test\_split**

[float, optional] Fraction of events reserved for testing. Default value: 0.2.

**validation\_split**

[float, optional] Fraction of weighted events reserved for validation. Default value: 0.2.

**gradients**

[{"all", "theta", "nu"}, optional] Which gradients to calculate. Default value: "all".

**batch\_size**

[int, optional] Size of the batches of events that are loaded into memory at the same time. Default value: 100000.

**generated\_close\_to**

[None or ndarray, optional] If not None, only events originally generated from the closest benchmark to this parameter point will be used. Default value : None.

**Returns****xsecs\_gradients**[ndarray] Calculated cross section gradients in pb with shape  $(n\_gradients,)$ .**xsecs**(*thetas=None, nus=None, partition='all', test\_split=0.2, validation\_split=0.2, include\_nuisance\_benchmarks=True, batch\_size=100000, generated\_close\_to=None*)

Returns the total cross sections for benchmarks or parameter points.

**Parameters****thetas**

[None or list of (ndarray or str), optional] If None, the function returns all benchmark cross

sections. Otherwise, it returns the cross sections for a series of parameter points that are either given by their benchmark name (as a str), their benchmark index (as an int), or their parameter value (as an ndarray, using morphing). Default value: None.

**nus**

[None or list of (None or ndarray), optional] If None, the nuisance parameters are set to their nominal values (0), i.e. no systematics are taken into account. Otherwise, the list has to have the same number of elements as thetas, and each entry can specify nuisance parameters at nominal value (None) or a value of the nuisance parameters (ndarray).

**partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default: "all".

**test\_split**

[float, optional] Fraction of events reserved for testing. Default value: 0.2.

**validation\_split**

[float, optional] Fraction of weighted events reserved for validation. Default value: 0.2.

**include\_nuisance\_benchmarks**

[bool, optional] Whether to include nuisance benchmarks if thetas is None. Default value: True.

**batch\_size**

[int, optional] Size of the batches of events that are loaded into memory at the same time. Default value: 100000.

**generated\_close\_to**

[None or ndarray, optional] If not None, only events originally generated from the closest benchmark to this parameter point will be used. Default value : None.

**Returns**

**xsecs**

[ndarray] Calculated cross sections in pb.

**xsec\_uncertainties**

[ndarray] Cross-section uncertainties in pb. Basically calculated as  $\text{sum}(\text{weights}^{**2})^{**0.5}$ .

## 6.3 Module contents

## MADMINER.CORE PACKAGE

### 7.1 Submodules

### 7.2 `madminer.core.madminer` module

**class** `madminer.core.madminer.MadMiner`

Bases: `object`

The central class to manage parameter spaces, benchmarks, and the generation of events through MadGraph and Pythia.

An instance of this class is the starting point of most MadMiner applications. It is typically used in four steps:

- Defining the parameter space through *MadMiner.add\_parameter*
- Defining the benchmarks, i.e. the points at which the squared matrix elements will be evaluated in MadGraph, with *MadMiner.add\_benchmark()* or, if operator morphing is used, with *MadMiner.set\_benchmarks\_from\_morphing()*
- Saving this setup with *MadMiner.save()* (it can be loaded in a new instance with *MadMiner.load()*)
- Running MadGraph and Pythia with the appropriate settings with *MadMiner.run()* or *MadMiner.run\_multiple()* (the latter allows the user to combine runs from multiple run cards and sampling points)

Please see the tutorial for a hands-on introduction to its methods.

## Methods

<code>add_benchmark</code> (parameter_values[, ...])	Manually adds an individual benchmark, that is, a parameter point that will be evaluated by MadGraph.
<code>add_parameter</code> (lha_block, lha_id[, ...])	Adds an individual parameter.
<code>add_systematics</code> (effect[, systematic_name, ...])	
<b>Parameters</b>	
<code>finite_differences</code> ([epsilon])	Adds benchmarks so that the score can be computed from finite differences
<code>load</code> (filename[, disable_morphing])	Loads MadMiner setup from a file.
<code>reweight_existing_sample</code> (...[, ...])	High-level function that adds the weights required for MadMiner to an existing sample.
<code>run</code> (mg_directory, proc_card_file, ...[, ...])	High-level function that creates the the MadGraph process, all required cards, and prepares or runs the event generation for one combination of cards.
<code>run_multiple</code> (mg_directory, proc_card_file, ...)	High-level function that creates the the MadGraph process, all required cards, and prepares or runs the event generation for multiple combinations of run_cards or importance samplings ( <i>sample_benchmarks</i> ).
<code>save</code> (filename)	Saves MadMiner setup into a file.
<code>set_benchmarks</code> (benchmarks[, verbose])	Manually sets all benchmarks, that is, parameter points that will be evaluated by MadGraph.
<code>set_morphing</code> ([max_overall_power, n_bases, ...])	Sets up the morphing environment.
<code>set_parameters</code> (parameters)	Manually sets all parameters, overwriting previously added parameters.

**add\_benchmark**(parameter\_values: Dict[str, float], benchmark\_name: Optional[str] = None, verbose: float = True)

Manually adds an individual benchmark, that is, a parameter point that will be evaluated by MadGraph.

### Parameters

#### parameter\_values

[dict] The keys of this dict should be the parameter names and the values the corresponding parameter values.

#### benchmark\_name

[str or None, optional] Name of benchmark. If None, a default name is used. Default value: None.

#### verbose

[bool, optional] If True, prints output about each benchmark. Default value: True.

### Returns

None

### Raises

#### RuntimeError

If a benchmark with the same name already exists, if parameter\_values is not a dict, or if a key of parameter\_values does not correspond to a defined parameter.

**add\_parameter**(lha\_block, lha\_id, parameter\_name=None, param\_card\_transform=None, morphing\_max\_power=2, parameter\_range=(0.0, 1.0))



Adds an individual parameter.

#### Parameters

##### **lha\_block**

[str] The name of the LHA block as used in the param\_card. Case-sensitive.

##### **lha\_id**

[int] The LHA id as used in the param\_card.

##### **parameter\_name**

[str or None] An internal name for the parameter. If None, a the default 'benchmark\_i' is used.

##### **morphing\_max\_power**

[int] The maximal power with which this parameter contributes to the squared matrix element of the process of interest. Typically at tree level, this maximal number is 2 for parameters that affect one vertex (e.g. only production or only decay of a particle), and 4 for parameters that affect two vertices (e.g. production and decay). Default value: 2.

##### **param\_card\_transform**

[None or str] Represents a one-parameter function mapping the parameter ("*theta*") to the value that should be written in the parameter cards. This str is parsed by Python's *eval()* function, and "*theta*" is parsed as the parameter value. Default value: None.

##### **parameter\_range**

[tuple of float] The range of parameter values of primary interest. Only affects the basis optimization. Default value: (0., 1.).

#### Returns

None

**add\_systematics**(*effect*, *systematic\_name*=None, *norm\_variation*=1.1, *scale*='mu', *scale\_variations*=(0.5, 1.0, 2.0), *pdf\_variation*='CT10')

#### Parameters

##### **effect**

[{"norm", "scale", "pdf"}] Type of the nuisance parameter. If "norm", it will affect the overall normalization of one or multiple samples in the process. If "scale", the nuisance parameter effect will be determined by varying factorization or regularization scales (depending on *scale\_variation* and *scales*). If "pdf", the effect of the nuisance parameters will be determined by varying the PDF used.

##### **systematic\_name**

[None or str, optional]

##### **scale**

[{"mu", "mur", "muf"}, optional] If type is "scale", this sets whether only the regularization scale ("mur"), only the factorization scale ("muf"), or both simultaneously ("mu") are varied. Default value: "mu".

##### **norm\_variation**

[float, optional] If type is "norm", this sets the relative effect of the nuisance parameter on the cross section at the "plus 1 sigma" variation. 1.1 corresponds to a 10% increase, 0.9 to a 10% decrease relative to the nominal cross section. Default value: 1.1.

##### **scale\_variations**

[tuple of float, optional] If type is "scale", this sets how the regularization and / or factor-

ization scales are varied. A tuple like (0.5, 1.0, 2.0) specifies the factors with which they are varied. Default value: (0.5, 1.0, 2.0).

#### **pdf\_variation**

[str, optional] If type is “pdf”, defines the PDF set for the variation. The option is passed along to the *-pdf* option of MadGraph’s systematics module. See <https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/Systematics> for a list. The option “CT10” would, as an example, run over all the eigenvectors of the CTEQ10 set. Default value: “CT10”.

#### **Returns**

None

#### **finite\_differences(*epsilon=0.01*)**

Adds benchmarks so that the score can be computed from finite differences

Don’t add any more benchmarks or parameters after calling this!

#### **load(*filename, disable\_morphing=False*)**

Loads MadMiner setup from a file. All parameters, benchmarks, and morphing settings are overwritten. See *save* for more details.

#### **Parameters**

##### **filename**

[str] Path to the MadMiner file.

##### **disable\_morphing**

[bool, optional] If True, the morphing setup is not loaded from the file. Default value: False.

#### **Returns**

None

#### **reweight\_existing\_sample(*mg\_process\_directory, run\_name, param\_card\_template\_file, sample\_benchmark, reweight\_benchmarks=None, only\_prepare\_script=False, log\_directory=None, initial\_command=None*)**

High-level function that adds the weights required for MadMiner to an existing sample.

If *only\_prepare\_scripts=True*, the event generation is not run directly, but a bash script is created in *<process\_folder>/madminer/run.sh* that will start the event generation with the correct settings.

Currently does not support adding systematics.

#### **Parameters**

##### **mg\_process\_directory**

[str] Path to the MG process directory. If None, MadMiner uses *./MG\_process*.

##### **run\_name**

[str] Run name.

##### **param\_card\_template\_file**

[str] Path to a param card that will be used as template to create the appropriate param cards for these runs.

##### **sample\_benchmark**

[str] The name of the benchmark used to generate this sample.

##### **reweight\_benchmarks**

[list of str or None] Lists the names of benchmarks to which the sample should be reweighted. If None, all benchmarks (except *sample\_benchmarks*) are used.

**only\_prepare\_script**

[bool, optional] If True, the event generation is not started, but instead a run.sh script is created in the process directory. Default value: False.

**log\_directory**

[str or None, optional] Directory for log files with the MadGraph output. If None, ./logs is used. Default value: None.

**initial\_command**

[str or None, optional] Initial shell commands that have to be executed before MG is run (e.g. to load a virtual environment). Default value: None.

**Returns**

None

**run**(mg\_directory, proc\_card\_file, param\_card\_template\_file, run\_card\_file=None, mg\_process\_directory=None, pythia8\_card\_file=None, configuration\_file=None, sample\_benchmark=None, is\_background=False, only\_prepare\_script=False, ufo\_model\_directory=None, log\_directory=None, temp\_directory=None, initial\_command=None, systematics=None, order='LO', python\_executable=None)

High-level function that creates the the MadGraph process, all required cards, and prepares or runs the event generation for one combination of cards.

If *only\_prepare\_scripts=True*, the event generation is not run directly, but a bash script is created in *<process\_folder>/madminer/run.sh* that will start the event generation with the correct settings.

High-level function that creates the the MadGraph process, all required cards, and prepares or runs the event generation for multiple combinations of run\_cards or importance samplings (*sample\_benchmarks*).

If *only\_prepare\_scripts=True*, the event generation is not run directly, but a bash script is created in *<process\_folder>/madminer/run.sh* that will start the event generation with the correct settings.

**Parameters**

**mg\_directory**

[str] Path to the MadGraph 5 base directory.

**proc\_card\_file**

[str] Path to the process card that tells MadGraph how to generate the process.

**param\_card\_template\_file**

[str] Path to a param card that will be used as template to create the appropriate param cards for these runs.

**run\_card\_file**

[str] Paths to the MadGraph run card. If None, the default run\_card is used.

**mg\_process\_directory**

[str or None, optional] Path to the MG process directory. If None, MadMiner uses ./MG\_process. Default value: None.

**pythia8\_card\_file**

[str or None, optional] Path to the MadGraph Pythia8 card. If None, the card present in the process folder is used. Default value: None.

**configuration\_file**

[str, optional] Path to the MadGraph me5\_configuration card. If None, the card present in the process folder is used. Default value: None.

**sample\_benchmark**

[list of str or None, optional] Lists the names of benchmarks that should be used to sample

events. A different sampling does not change the expected differential cross sections, but will change which regions of phase space have many events (small variance) or few events (high variance). If None, the benchmark added first is used. Default value: None.

**is\_background**

[bool, optional] Should be True for background processes, i.e. process in which the differential cross section does not depend on the parameters (i.e. is the same for all benchmarks). In this case, no reweighting is run, which can substantially speed up the event generation. Default value: False.

**only\_prepare\_script**

[bool, optional] If True, the event generation is not started, but instead a run.sh script is created in the process directory. Default value: False.

**ufo\_model\_directory**

[str or None, optional] Path to an UFO model directory that should be used, but is not yet installed in mg\_directory/models. The model will be copied to the MadGraph model directory before the process directory is generated. (Default value = None).

**log\_directory**

[str or None, optional] Directory for log files with the MadGraph output. If None, ./logs is used. Default value: None.

**temp\_directory**

[str or None, optional] Path to a temporary directory. If None, a system default is used. Default value: None.

**initial\_command**

[str or None, optional] Initial shell commands that have to be executed before MG is run (e.g. to load a virtual environment). Default value: None.

**systematics**

[None or list of str, optional] If list of str, defines which systematics are used for this run.

**order**

['LO' or 'NLO', optional] Differentiates between LO and NLO order runs. Minor changes to writing, reading and naming cards. Default value: 'LO'

**python\_executable**

[None or str, optional] Provides a path to the Python executable that should be used to call MadMiner. Default: None.

**Returns**

None

**run\_multiple**(mg\_directory, proc\_card\_file, param\_card\_template\_file, run\_card\_files, mg\_process\_directory=None, pythia8\_card\_file=None, configuration\_file=None, sample\_benchmarks=None, is\_background=False, only\_prepare\_script=False, ufo\_model\_directory=None, log\_directory=None, temp\_directory=None, initial\_command=None, systematics=None, order='LO', python\_executable=None)

High-level function that creates the the MadGraph process, all required cards, and prepares or runs the event generation for multiple combinations of run\_cards or importance samplings (*sample\_benchmarks*).

If *only\_prepare\_scripts=True*, the event generation is not run directly, but a bash script is created in *<process\_folder>/madminer/run.sh* that will start the event generation with the correct settings.

**Parameters**

**mg\_directory**

[str] Path to the MadGraph 5 base directory.

**proc\_card\_file**

[str] Path to the process card that tells MadGraph how to generate the process.

**param\_card\_template\_file**

[str] Path to a param card that will be used as template to create the appropriate param cards for these runs.

**run\_card\_files**

[list of str] Paths to the MadGraph run card.

**mg\_process\_directory**

[str or None, optional] Path to the MG process directory. If None, MadMiner uses ./MG\_process. Default value: None.

**pythia8\_card\_file**

[str, optional] Path to the MadGraph Pythia8 card. If None, the card present in the process folder is used. Default value: None.

**configuration\_file**

[str, optional] Path to the MadGraph me5\_configuration card. If None, the card present in the process folder is used. Default value: None.

**sample\_benchmarks**

[list of str or None, optional] Lists the names of benchmarks that should be used to sample events. A different sampling does not change the expected differential cross sections, but will change which regions of phase space have many events (small variance) or few events (high variance). If None, a run is started for each of the benchmarks, which should map out all regions of phase space well. Default value: None.

**is\_background**

[bool, optional] Should be True for background processes, i.e. process in which the differential cross section does not depend on the parameters (i.e. is the same for all benchmarks). In this case, no reweighting is run, which can substantially speed up the event generation. Default value: False.

**only\_prepare\_script**

[bool, optional] If True, the event generation is not started, but instead a run.sh script is created in the process directory. Default value: False.

**ufo\_model\_directory**

[str or None, optional] Path to an UFO model directory that should be used, but is not yet installed in mg\_directory/models. The model will be copied to the MadGraph model directory before the process directory is generated. (Default value = None)

**log\_directory**

[str or None, optional] Directory for log files with the MadGraph output. If None, ./logs is used. Default value: None.

**temp\_directory**

[str or None, optional] Path to a temporary directory. If None, a system default is used. Default value: None.

**initial\_command**

[str or None, optional] Initial shell commands that have to be executed before MG is run (e.g. to load a virtual environment). If not specified and *python2\_override* is True, it adds the user-installed Python2 binaries to the PATH. Default value: None.

**systematics**

[None or list of str, optional] If list of str, defines which systematics are used for these runs.

**order**

['LO' or 'NLO', optional] Differentiates between LO and NLO order runs. Minor changes to writing, reading and naming cards. Default value: 'LO'

**python\_executable**

[None or str, optional] Provides a path to the Python executable that should be used to call MadMiner. Default: None.

**Returns**

None

**save(filename)**

Saves MadMiner setup into a file.

The file format follows the HDF5 standard. The saved information includes:

- the parameter definitions,
- the benchmark points,
- the systematics setup (if defined), and
- the morphing setup (if defined).

This file is an important input to later stages in the analysis chain, including the processing of generated events, extraction of training samples, and calculation of Fisher information matrices. In these downstream tasks, additional information will be written to the MadMiner file, including the observations and event weights.

**Parameters**

**filename**

[str] Path to the MadMiner file.

**Returns**

None

**set\_benchmarks(benchmarks: Union[Dict[str, dict], List[dict]], verbose: bool = True)**

Manually sets all benchmarks, that is, parameter points that will be evaluated by MadGraph. Calling this function overwrites all previously defined benchmarks.

**Parameters**

**benchmarks**

[dict or list] Specifies all benchmarks. If None, all benchmarks are reset. If dict, the keys are the benchmark names and the values the Benchmark instances. If list, the entries are dicts {parameter\_name:value} (and the benchmark names are chosen automatically). Default value: None.

**verbose**

[bool, optional] If True, prints output about each benchmark. Default value: True.

**Returns**

None

**set\_morphing(max\_overall\_power=4, n\_bases=1, include\_existing\_benchmarks=True, n\_trials=100, n\_test\_thetas=100)**

Sets up the morphing environment.

Sets benchmarks, i.e. parameter points that will be evaluated by MadGraph, for a morphing algorithm, and calculates all information required for morphing. Morphing is a technique that allows MadMax to

infer the full probability distribution  $p(x_i | \theta)$  for each simulated event  $x_i$  and any  $\theta$ , not just the benchmarks.

The morphing basis is optimized with respect to the expected mean squared morphing weights over the parameter region of interest. If `keep_existing_benchmarks=True`, benchmarks defined previously will be incorporated in the morphing basis and only the remaining basis points will be optimized.

Note that any subsequent call to `set_benchmarks` or `add_benchmark` will overwrite the morphing setup. The correct order is therefore to manually define benchmarks first, using `set_benchmarks` or `add_benchmark`, and then to create the morphing setup and complete the basis by calling `set_benchmarks_from_morphing(keep_existing_benchmarks=True)`.

### Parameters

#### **max\_overall\_power**

[int, optional] The maximal sum of powers of all parameters contributing to the squared matrix element. Typically, if parameters can affect the couplings at  $n$  vertices, this number is  $2n$ . Default value: 4.

#### **n\_bases**

[int, optional] The number of morphing bases generated. If `n_bases > 1`, multiple bases are combined, and the weights for each basis are reduced by a factor  $1 / n\_bases$ . Currently only the default choice of 1 is fully implemented. Do not use any other value for now. Default value: 1.

#### **include\_existing\_benchmarks**

[bool, optional] If True, the previously defined benchmarks are included in the morphing basis. In that case, the number of free parameters in the optimization routine is reduced. If False, the existing benchmarks will still be simulated, but are not part of the morphing routine. Default value: True.

#### **n\_trials**

[int, optional] Number of random basis configurations tested in the optimization procedure. A larger number will increase the run time of the optimization, but lead to better results. Default value: 100.

#### **n\_test\_thetas**

[int, optional] Number of random parameter points used to evaluate the expected mean squared morphing weights. A larger number will increase the run time of the optimization, but lead to better results. Default value: 100.

### Returns

None

**set\_parameters**(*parameters*: Union[Dict[str, AnalysisParameter], List[tuple]])

Manually sets all parameters, overwriting previously added parameters.

### Parameters

#### **parameters**

[dict or list] If `parameters` is a dict, the keys should be str and give the parameter names, and the values are `AnalysisParameter` model instances. If `parameters` is a list, the items should be tuples of the form (LHA\_block, LHA\_ID).

### Returns

None

## 7.3 Module contents



## MADMINER.DELPHES PACKAGE

### 8.1 Submodules

### 8.2 `madminer.delphes.delphes_reader` module

**class** `madminer.delphes.delphes_reader.DelphesReader(filename)`

Bases: `object`

Detector simulation with Delphes and simple calculation of observables.

After setting up the parameter space and benchmarks and running MadGraph and Pythia, all of which is organized in the `madminer.core.MadMiner` class, the next steps are the simulation of detector effects and the calculation of observables. Different tools can be used for these tasks, please feel free to implement the detector simulation and analysis routine of your choice.

This class provides an example implementation based on Delphes. Its workflow consists of the following steps:

- Initializing the class with the filename of a MadMiner HDF5 file (the output of `madminer.core.MadMiner.save()`)
- Adding one or multiple event samples produced by MadGraph and Pythia in `DelphesProcessor.add_sample()`.
- Running Delphes on the samples that require it through `DelphesProcessor.run_delphes()`.
- Optionally, acceptance cuts for all visible particles can be defined with `DelphesProcessor.set_acceptance()`.
- Defining observables through `DelphesProcessor.add_observable()` or `DelphesProcessor.add_observable_from_function()`. A simple set of default observables is provided in `DelphesProcessor.add_default_observables()`
- Optionally, cuts can be set with `DelphesProcessor.add_cut()`
- Calculating the observables from the Delphes ROOT files with `DelphesProcessor.analyse_delphes_samples()`
- Saving the results with `DelphesProcessor.save()`

Please see the tutorial for a detailed walk-through.

#### Parameters

##### **filename**

[str or None, optional] Path to MadMiner file (the output of `madminer.core.MadMiner.save()`). Default value: None.

## Methods

<code>add_cut</code> (definition[, required])	Adds a cut as a string that can be parsed by Python's <code>eval()</code> function and returns a bool.
<code>add_default_observables</code> ([n_leptons_max, ...])	Adds a set of simple standard observables: the four-momenta (parameterized as E, pT, eta, phi) of the hardest visible particles, and the missing transverse energy.
<code>add_observable</code> (name, definition[, required, ...])	Adds an observable as a string that can be parsed by Python's <code>eval()</code> function.
<code>add_observable_from_function</code> (name, fn[, ...])	Adds an observable defined through a function.
<code>add_sample</code> (hepmc_filename, ...[, ...])	Adds a sample of simulated events.
<code>analyse_delphes_samples</code> ([generator_truth, ...])	Main function that parses the Delphes samples (ROOT files), checks acceptance and cuts, and extracts the observables and weights.
<code>reset_cuts</code> ()	Resets all cuts.
<code>reset_observables</code> ()	Resets all observables.
<code>run_delphes</code> (delphes_directory, delphes_card)	Runs the fast detector simulation Delphes on all HepMC samples added so far for which it hasn't been run yet.
<code>save</code> (filename_out[, shuffle])	Saves the observable definitions, observable values, and event weights in a MadMiner file.
<code>set_acceptance</code> ([pt_min_e, pt_min_mu, ...])	Sets acceptance cuts for all visible particles.

**add\_cut** (definition, required=False)

Adds a cut as a string that can be parsed by Python's `eval()` function and returns a bool.

### Parameters

#### definition

[str] An expression that can be parsed by Python's `eval()` function and returns a bool: True for the event to pass this cut, False for it to be rejected. In the definition, all visible particles can be used: *e*, *mu*, *j*, *a*, and *l* provide lists of electrons, muons, jets, photons, and leptons (electrons and muons combined), in each case sorted by descending transverse momentum. *met* provides a missing ET object. *visible* and *all* provide access to the sum of all visible particles and the sum of all visible particles plus MET, respectively. In addition, *MadMinerParticle* have properties *charge* and *pdg\_id*, which return the charge in units of elementary charges (i.e. an electron has `e[0].charge = -1.`), and the PDG particle ID. For instance, "`len(e) >= 2`" requires at least two electrons passing the acceptance cuts, while "`mu[0].charge > 0.`" specifies that the hardest muon is positively charged.

#### required

[bool, optional] Whether the cut is passed if the observable cannot be parsed. Default value: False.

### Returns

None

**add\_default\_observables** (n\_leptons\_max=2, n\_photons\_max=2, n\_jets\_max=2, include\_met=True, include\_visible\_sum=True, include\_numbers=True, include\_charge=True)

Adds a set of simple standard observables: the four-momenta (parameterized as E, pT, eta, phi) of the hardest visible particles, and the missing transverse energy.

### Parameters

**n\_leptons\_max**

[int, optional] Number of hardest leptons for which the four-momenta are saved. Default value: 2.

**n\_photons\_max**

[int, optional] Number of hardest photons for which the four-momenta are saved. Default value: 2.

**n\_jets\_max**

[int, optional] Number of hardest jets for which the four-momenta are saved. Default value: 2.

**include\_met**

[bool, optional] Whether the missing energy observables are stored. Default value: True.

**include\_visible\_sum**

[bool, optional] Whether observables characterizing the sum of all particles are stored. Default value: True.

**include\_numbers**

[bool, optional] Whether the number of leptons, photons, and jets is saved as observable. Default value: True.

**include\_charge**

[bool, optional] Whether the lepton charge is saved as observable. Default value: True.

**Returns**

None

**add\_observable**(*name*, *definition*, *required=False*, *default=None*)

Adds an observable as a string that can be parsed by Python's *eval()* function.

**Parameters****name**

[str] Name of the observable. Since this name will be used in *eval()* calls for cuts, this should not contain spaces or special characters.

**definition**

[str] An expression that can be parsed by Python's *eval()* function. As objects, the visible particles can be used: *e*, *mu*, *j*, *a*, and *l* provide lists of electrons, muons, jets, photons, and leptons (electrons and muons combined), in each case sorted by descending transverse momentum. *met* provides a missing ET object. *visible* and *all* provide access to the sum of all visible particles and the sum of all visible particles plus MET, respectively. In addition, *MadMinerParticle* have properties *charge* and *pdg\_id*, which return the charge in units of elementary charges (i.e. an electron has *e[0].charge = -1.*), and the PDG particle ID. For instance, "*abs(j[0].phi - j[1].phi)*" defines the azimuthal angle between the two hardest jets.

**required**

[bool, optional] Whether the observable is required. If True, an event will only be retained if this observable is successfully parsed. For instance, any observable involving "*j[1]*" will only be parsed if there are at least two jets passing the acceptance cuts. Default value: False.

**default**

[float or None, optional] If *required=False*, this is the placeholder value for observables that cannot be parsed. None is replaced with *np.nan*. Default value: None.

**Returns**

None

**add\_observable\_from\_function**(*name*, *fn*, *required=False*, *default=None*)

Adds an observable defined through a function.

#### Parameters

##### **name**

[str] Name of the observable. Since this name will be used in *eval()* calls for cuts, this should not contain spaces or special characters.

##### **fn**

[function] A function with signature *observable(leptons, photons, jets, met)* where the input arguments are lists of MadMinerParticle instances and a float is returned. The function should raise a *RuntimeError* to signal that it is not defined.

##### **required**

[bool, optional] Whether the observable is required. If True, an event will only be retained if this observable is successfully parsed. For instance, any observable involving “*j[1]*” will only be parsed if there are at least two jets passing the acceptance cuts. Default value: False.

##### **default**

[float or None, optional] If *required=False*, this is the placeholder value for observables that cannot be parsed. None is replaced with *np.nan*. Default value: None.

#### Returns

None

**add\_sample**(*hepmc\_filename*, *sampled\_from\_benchmark*, *is\_background=False*, *delphes\_filename=None*, *lhe\_filename=None*, *k\_factor=1.0*, *weights='lhe'*, *systematics=None*)

Adds a sample of simulated events. A HepMC file (from Pythia) has to be provided always, since some relevant information is only stored in this file. The user can optionally provide a Delphes file, in this case *run\_delphes()* does not have to be called.

By default, the weights are read out from the Delphes file and their names from the HepMC file. There are some issues with current MadGraph versions that lead to Pythia not storing the weights. As work-around, MadMiner supports reading weights from the LHE file (the observables still come from the Delphes file). To enable this, use *weights='lhe'*.

#### Parameters

##### **hepmc\_filename**

[str] Path to the HepMC event file (with extension ‘.hepmc’ or ‘.hepmc.gz’).

##### **sampled\_from\_benchmark**

[str] Name of the benchmark that was used for sampling in this event file (the keyword *sample\_benchmark* of *madminer.core.MadMiner.run()*).

##### **is\_background**

[bool, optional] Whether the sample is a background sample (i.e. without benchmark reweighting).

##### **delphes\_filename**

[str or None, optional] Path to the Delphes event file (with extension ‘.root’). If None, the user has to call *run\_delphes()*, which will create this file. Default value: None.

##### **lhe\_filename**

[None or str, optional] Path to the LHE event file (with extension ‘.lhe’ or ‘.lhe.gz’). This is only needed if *weights* is “*lhe*”.

### **k\_factor**

[float, optional] Multiplies the cross sections found in the sample. Default value: 1.

### **weights**

[{"delphes", "lhe"}, optional] If "delphes", the weights are read out from the Delphes ROOT file, and their names are taken from the HepMC file. If "lhe" (and lhe\_filename is not None), the weights are taken from the LHE file (and matched with the observables from the Delphes ROOT file). The "delphes" behaviour is generally better as it minimizes the risk of mismatching observables and weights, but for some MadGraph and Delphes versions there are issues with weights not being saved in the HepMC and Delphes ROOT files. In this case, setting weights to "lhe" and providing the unweighted LHE file from MadGraph may be an easy fix. Default value: "lhe".

### **systematics**

[None or list of str, optional] List of systematics associated with this sample. Default value: None.

### **Returns**

None

**analyse\_delphes\_samples**(*generator\_truth=False, delete\_delphes\_files=False, reference\_benchmark=None, parse\_lhe\_events\_as\_xml=True*)

Main function that parses the Delphes samples (ROOT files), checks acceptance and cuts, and extracts the observables and weights.

### **Parameters**

#### **generator\_truth**

[bool, optional] If True, the generator truth information (as given out by Pythia) will be parsed. Detector resolution or efficiency effects will not be taken into account.

#### **delete\_delphes\_files**

[bool, optional] If True, the Delphes ROOT files will be deleted after extracting the information from them. Default value: False.

#### **reference\_benchmark**

[str or None, optional] The weights at the nuisance benchmarks will be rescaled to some reference theta benchmark:  $dsigma(x|theta_{sampling}(x),nu) \rightarrow dsigma(x|theta_{ref},nu) = dsigma(x|theta_{sampling}(x),nu) * dsigma(x|theta_{ref},0) / dsigma(x|theta_{sampling}(x),0)$ . This sets the name of the reference benchmark. If None, the first one will be used. Default value: None.

#### **parse\_lhe\_events\_as\_xml**

[bool, optional] Decides whether the LHE events are parsed with an XML parser (more robust, but slower) or a text parser (less robust, faster). Default value: True.

### **Returns**

None

### **reset\_cuts()**

Resets all cuts.

### **reset\_observables()**

Resets all observables.

### **run\_delphes**(*delphes\_directory, delphes\_card, initial\_command=None, log\_file=None*)

Runs the fast detector simulation Delphes on all HepMC samples added so far for which it hasn't been run yet.

### Parameters

#### **delphes\_directory**

[str] Path to the Delphes directory.

#### **delphes\_card**

[str] Path to a Delphes card.

#### **initial\_command**

[str or None, optional] Initial bash commands that have to be executed before Delphes is run (e.g. to load the correct virtual environment). Default value: None.

#### **log\_file**

[str or None, optional] Path to log file in which the Delphes output is saved. Default value: None.

### Returns

None

**save**(*filename\_out*, *shuffle*=True)

Saves the observable definitions, observable values, and event weights in a MadMiner file. The parameter, benchmark, and morphing setup is copied from the file provided during initialization. Nuisance benchmarks found in the HepMC file are added.

### Parameters

#### **filename\_out**

[str] Path to where the results should be saved.

#### **shuffle**

[bool, optional] If True, events are shuffled before being saved. That's important when there are multiple distinct samples (e.g. signal and background). Default value: True.

### Returns

None

**set\_acceptance**(*pt\_min\_e*=None, *pt\_min\_mu*=None, *pt\_min\_a*=None, *pt\_min\_j*=None, *eta\_max\_e*=None, *eta\_max\_mu*=None, *eta\_max\_a*=None, *eta\_max\_j*=None)

Sets acceptance cuts for all visible particles. These are taken into account before observables and cuts are calculated.

### Parameters

#### **pt\_min\_e**

[float or None, optional] Minimum electron transverse momentum in GeV. None means no acceptance cut. Default value: None.

#### **pt\_min\_mu**

[float or None, optional] Minimum muon transverse momentum in GeV. None means no acceptance cut. Default value: None.

#### **pt\_min\_a**

[float or None, optional] Minimum photon transverse momentum in GeV. None means no acceptance cut. Default value: None.

#### **pt\_min\_j**

[float or None, optional] Minimum jet transverse momentum in GeV. None means no acceptance cut. Default value: None.

**eta\_max\_e**

[float or None, optional] Maximum absolute electron pseudorapidity. None means no acceptance cut. Default value: None.

**eta\_max\_mu**

[float or None, optional] Maximum absolute muon pseudorapidity. None means no acceptance cut. Default value: None.

**eta\_max\_a**

[float or None, optional] Maximum absolute photon pseudorapidity. None means no acceptance cut. Default value: None.

**eta\_max\_j**

[float or None, optional] Maximum absolute jet pseudorapidity. None means no acceptance cut. Default value: None.

**Returns**

None

## 8.3 Module contents





## MADMINER.FISHERINFORMATION PACKAGE

### 9.1 Submodules

### 9.2 madminer.fisherinformation.geometry module

**class** madminer.fisherinformation.geometry.InformationGeometry

Bases: object

Functions to calculate limits using Information Geometry.

After initializing the *InformationGeometry* class, a Fisher Information needs to be provided using one of the following functions

- *InformationGeometry.information\_from\_formula()* defines the Fisher Information explicitly as function of the theory parameters *theta*.
- *InformationGeometry.information\_from\_grid()* loads a grid of Fisher Information which is then interpolated.

Using information geometrical methods, the function *InformationGeometry.distance\_contours()* then calculates the distance contours and equivalently the p-values throughout parameter space.

#### Methods

<i>distance_contours</i> (theta0, grid_ranges, ...)	Finds the distance values from the point theta0 and the corresponding p-value within the parameter space bounded by <i>grid_ranges</i> .
<i>find_trajectory</i> (theta0, dtheta0, limits[, ...])	Finds the geodesic trajectory starting at a parameter point theta0 going in the initial direction dtheta0.
<i>information_from_formula</i> (formula, dimension)	Explicitly defines the Fisher Information as function of the theory parameter <i>theta</i> through a formula that can be evaluated using <i>eval()</i> .
<i>information_from_grid</i> (theta_grid, ...[, ...])	Loads a grid of coordinates and corresponding Fisher Information, which is then interpolated.

**distance\_contours**(theta0, grid\_ranges, grid\_resolutions, stepsize=None, ntrajectories=None, continuous\_sampling=False, return\_trajectories=False)

Finds the distance values from the point theta0 and the corresponding p-value within the parameter space bounded by *grid\_ranges*.

#### Parameters

**theta0**

[ndarray] Parameter point *theta0* at which the geodesic trajectory starts.

**grid\_ranges**

[list of (tuple of float)] Specifies the boundaries of the parameter grid in which the trajectory is evaluated. It should be  $[[min, max], [min, max], \dots, [min, max]]$ , where the list goes over all parameters and *min* and *max* are float.

**grid\_resolutions**

[list of int] Resolution of the parameter space grid on which the p-values are evaluated. The individual entries specify the number of points along each parameter individually.

**stepsize**

[float or None, optional] Maximal stepsize  $|\Delta \theta|$  during numerical integration in parameter space. If None,  $stepsize = \min([(max-min)/20 \text{ for } (min,max) \text{ in } grid\_ranges])$ . Default: None

**ntrajectories**

[int or None, optional] Number of sampled trajectories. If None,  $ntrajectories = 20$  times the number of dimensions. Default: None

**continous\_sampling**

[bool, optional] If *n\_dimension* is 2, the trajectories are sampled continously in the angular direction. Default: False

**return\_trajectories**

[bool, optional] Returns the trajectories (parameter points and distances). Default: False

**Returns**

**theta\_grid**

[ndarray] Parameter points at which the p-values are evaluated with shape  $(n\_grid\_points, n\_dimension)$ .

**p\_values**

[ndarray] Observed p-values for each parameter point on the grid, with shape  $(n\_grid\_points,)$ .

**p\_values**

[ndarray] Interpolated distance from *theta0* for each parameter point on the grid, with shape  $(n\_grid\_points,)$ .

**(list\_of\_theta, list\_of\_distance)**

[(ndarray,ndarray)] Only returned if *return\_trajectories* is True. List of parameter points *theta*  $(n\_points, n\_dimension)$  and List of distances from the staring point *theta0*  $(n\_points,)$ .

**find\_trajectory(theta0, dtheta0, limits, stepsize=1)**

Finds the geodesic trajectory starting at a parameter point *theta0* going in the initial direction *dtheta0*.

**Parameters**

**theta0**

[ndarray] Parameter point *theta0* at which the geodesic trajectory starts.

**dtheta0**

[ndarray] Initial direction *dtheta0* of the geodesic

**limits**

[list of (tuple of float)] Specifies the boundaries of the parameter grid in which the trajectory is evaluated. It should be  $[[min, max], [min, max], \dots, [min, max]]$ , where the list goes over all parameters and *min* and *max* are float.

**stepsize**

[int, optional] Maximal stepsize  $|\Delta \theta|$  during numerical integration in parameter space. \$Default: 1

**Returns****list\_of\_theta**

[ndarray] List of parameter points  $\theta$  ( $n\_points$ ,  $n\_dimension$ ).

**list\_of\_distance**

[ndarray] List of distances from the starting point  $\theta_0$  ( $n\_points$ , ).

**information\_from\_formula**(*formula*, *dimension*)

Explicitly defines the Fisher Information as function of the theory parameter  $\theta$  through a formula that can be evaluated using *eval()*.

**Parameters****formula**

[str] Explicit definition of the Fisher Information as ndarray, which can be a function of the n-dimensional theory parameter  $\theta$ . Example: `formula="np.array([[1+theta[0],1],[1,2*theta[1]**2]])"`

**dimension**

[int] Dimensionality of the theory parameter space.

**information\_from\_grid**(*theta\_grid*, *fisherinformation\_grid*, *option*='smooth', *inverse*='exact')

Loads a grid of coordinates and corresponding Fisher Information, which is then interpolated.

**Parameters****theta\_grid**

[ndarray] List of parameter points  $\theta$  at which the Fisher information matrices  $I_{ij}(\theta)$  is evaluated. Shape ( $n\_gridpoints$ ,  $n\_dimension$ ).

**fisherinformation\_grid**

[ndarray] List of Fisher information matrices  $I_{ij}(\theta)$ . Shape ( $n\_gridpoints$ ,  $n\_dimension$ ,  $n\_dimension$ ).

**option**

[{"smooth", "linear"}] Defines if the Fisher Information is interpolated smoothly using the function `CloughTocher2DInterpolator()` or piecewise linear using `LinearNDInterpolator()`. Default = 'smooth'.

**inverse**

[{"exact", "interpolate"}] Defines if the inverse Fisher Information is obtained by either first interpolating the Fisher Information and then inverting it ("exact") or by first inverting the grid of Fisher Informations and then interpolating the inverse ("interpolate"). Default = 'exact'.

## 9.3 madminer.fisherinformation.information module

```
class madminer.fisherinformation.information.FisherInformation(filename, include_nuisance_parameters=True)
```

Bases: *DataAnalyzer*

Functions to calculate expected Fisher information matrices.

After initializing a *FisherInformation* instance with the filename of a MadMiner file, different information matrices can be calculated:

- *FisherInformation.truth\_information()* calculates the full truth-level Fisher information. This is the information in an idealized measurement where all parton-level particles with their charges, flavours, and four-momenta can be accessed with perfect accuracy.
- *FisherInformation.full\_information()* calculates the full Fisher information in realistic detector-level observations, estimated with neural networks. In addition to the MadMiner file, this requires a trained SALLY or SALLINO estimator as well as an unweighted evaluation sample.
- *FisherInformation.rate\_information()* calculates the Fisher information in the total cross section.
- *FisherInformation.histo\_information()* calculates the Fisher information in the histogram of one (parton-level or detector-level) observable.
- *FisherInformation.histo\_information\_2d()* calculates the Fisher information in a two-dimensional histogram of two (parton-level or detector-level) observables.
- *FisherInformation.histogram\_of\_information()* calculates the full truth-level Fisher information in different slices of one observable (the “distribution of the Fisher information”).

Finally, don’t forget that in the presence of nuisance parameters the constraint terms also affect the Fisher information. This term is given by *FisherInformation.calculate\_fisher\_information\_nuisance\_constraints()*.

### Parameters

#### **filename**

[str] Path to MadMiner file (for instance the output of *madminer.delphes.DelphesProcessor.save()*).

#### **include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

## Methods

<code>calculate_fisher_information_full_detector(theta)</code>	Calculates the full Fisher information in realistic detector-level observations, estimated with neural networks.
<code>calculate_fisher_information_full_truth(theta)</code>	Calculates the full Fisher information at parton / truth level.
<code>calculate_fisher_information_hist1d(theta, ...)</code>	Calculates the Fisher information in the one-dimensional histogram of an (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observable.
<code>calculate_fisher_information_hist2d(theta, ...)</code>	Calculates the Fisher information in a two-dimensional histogram of two (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observables.
<code>calculate_fisher_information_nuisance_constraints(theta)</code>	Builds the Fisher information term representing the Gaussian constraints on the nuisance parameters
<code>calculate_fisher_information_rate(theta, ...)</code>	Calculates the Fisher information in a measurement of the total cross section (without any kinematic information).
<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>full_information(theta, model_file[, ...])</code>	Calculates the full Fisher information in realistic detector-level observations, estimated with neural networks.
<code>histo_information(theta, luminosity, ...[, ...])</code>	Calculates the Fisher information in the one-dimensional histogram of an (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observable.
<code>histo_information_2d(theta, luminosity, ...)</code>	Calculates the Fisher information in a two-dimensional histogram of two (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observables.
<code>histogram_of_fisher_information(theta, ...)</code>	Calculates the full and rate-only Fisher information in slices of one observable.
<code>histogram_of_information(theta, observable, ...)</code>	Calculates the full and rate-only Fisher information in slices of one observable.
<code>histogram_of_sigma_dsigma(theta, observable, ...)</code>	Fills events into histograms and calculates the cross section and first derivative for each bin
<code>nuisance_constraint_information()</code>	Builds the Fisher information term representing the Gaussian constraints on the nuisance parameters
<code>rate_information(theta, luminosity[, cuts, ...])</code>	Calculates the Fisher information in a measurement of the total cross section (without any kinematic information).
<code>truth_information(theta[, luminosity, cuts, ...])</code>	Calculates the full Fisher information at parton / truth level.
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

```
calculate_fisher_information_full_detector(theta, model_file, unweighted_x_sample_file=None,  
                                           luminosity=300000.0, include_xsec_info=True,  
                                           mode='score', calculate_covariance=True,  
                                           batch_size=100000, test_split=0.2)
```

Calculates the full Fisher information in realistic detector-level observations, estimated with neural networks. In addition to the MadMiner file, this requires a trained SALLY or SALLINO estimator.

Nuisance parameter are taken into account automatically if the SALLY / SALLINO model was trained with them.

#### Parameters

##### **theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

##### **model\_file**

[str] Filename of a trained local score regression model that was trained on samples from *theta* (see *madminer.ml.Estimator*).

##### **unweighted\_x\_sample\_file**

[str or None] Filename of an unweighted x sample that is sampled according to *theta* and obeys the cuts (see *madminer.sampling.SampleAugmenter.extract\_samples\_train\_local()*). If None, the Fisher information is instead calculated on the full, weighted samples (the data in the MadMiner file). Default value: None.

##### **luminosity**

[float, optional] Luminosity in pb<sup>-1</sup>. Default value: 300000.

##### **include\_xsec\_info**

[bool, optional] Whether the rate information is included in the returned Fisher information. Default value: True.

##### **mode**

[{"score", "information"}, optional] How the ensemble uncertainty on the kinematic Fisher information is calculated. If mode is "information", the Fisher information for each estimator is calculated individually and only then are the sample mean and covariance calculated. If mode is "score", the sample mean is calculated for the score for each event. Default value: "score".

##### **calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: True.

##### **batch\_size**

[int, optional] Batch size. Default value: 100000.

##### **test\_split**

[float or None, optional] If *unweighted\_x\_sample\_file* is None, this determines the fraction of weighted events used for evaluation. If None, all events are used (this will probably include events used during training!). Default value: 0.2.

#### Returns

##### **fisher\_information**

[ndarray or list of ndarray] Estimated expected full detector-level Fisher information matrix with shape (*n\_parameters*, *n\_parameters*). If more than one value ensemble\_vote\_expectation\_weight is given, this is a list with results for all entries in ensemble\_vote\_expectation\_weight.

**fisher\_information\_uncertainty**

[ndarray or list of ndarray or None] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ . If more than one value `ensemble_vote_expectation_weight` is given, this is a list with results for all entries in `ensemble_vote_expectation_weight`.

**calculate\_fisher\_information\_full\_truth**(*theta*, *luminosity*=300000.0, *cuts*=None, *efficiency\_functions*=None, *include\_nuisance\_parameters*=True)

Calculates the full Fisher information at parton / truth level. This is the information in an idealized measurement where all parton-level particles with their charges, flavours, and four-momenta can be accessed with perfect accuracy, i.e. the latent variables  $z\_parton$  can be measured directly.

**Parameters****theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(theta)$  is evaluated.

**luminosity**

[float] Luminosity in  $\text{pb}^{-1}$ .

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

**Returns****fisher\_information**

[ndarray] Expected full truth-level Fisher information matrix with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

**calculate\_fisher\_information\_hist1d**(*theta*, *luminosity*, *observable*, *bins*, *histrange*=None, *cuts*=None, *efficiency\_functions*=None, *n\_events\_dynamic\_binning*=None)

Calculates the Fisher information in the one-dimensional histogram of an (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observable.

**Parameters****theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(theta)$  is evaluated.

**luminosity**

[float] Luminosity in  $\text{pb}^{-1}$ .

**observable**

[str] Expression for the observable to be histogrammed. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**bins**

[int or ndarray] If int: number of bins in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries (excluding overflow bins).

**histrange**

[tuple of float or None, optional] Minimum and maximum value of the histogram in the form (*min*, *max*). Overflow bins are always added. If None and bins is an int, variable-width bins with equal cross section are constructed automatically. Default value: None.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**n\_events\_dynamic\_binning**

[int or None, optional] Number of events used to calculate the dynamic binning (if *histrange* is None). If None, all events are used. Note that these events are not shuffled, so if the events in the MadMiner file are sorted, using a value different from None can cause issues. Default value: None.

**Returns**

**fisher\_information**

[ndarray] Expected Fisher information in the histogram with shape (*n\_parameters*, *n\_parameters*).

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape (*n\_parameters*, *n\_parameters*, *n\_parameters*, *n\_parameters*), calculated with plain Gaussian error propagation.

**calculate\_fisher\_information\_hist2d**(*theta*, *luminosity*, *observable1*, *bins1*, *observable2*, *bins2*, *histrange1*=None, *histrange2*=None, *cuts*=None, *efficiency\_functions*=None, *n\_events\_dynamic\_binning*=None)

Calculates the Fisher information in a two-dimensional histogram of two (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observables.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**luminosity**

[float] Luminosity in pb<sup>-1</sup>.

**observable1**

[str] Expression for the first observable to be histogrammed. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**bins1**

[int or ndarray] If int: number of bins along the first axis in the histogram in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries along the first axis in the histogram (excluding overflow bins).



**observable2**

[str] Expression for the first observable to be histogrammed. The str will be parsed by Python's `eval()` function and can use the names of the observables in the MadMiner files.

**bins2**

[int or ndarray] If int: number of bins along the second axis in the histogram in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries along the second axis in the histogram (excluding overflow bins).

**histrange1**

[tuple of float or None, optional] Minimum and maximum value of the first axis of the histogram in the form  $(min, max)$ . Overflow bins are always added. If None, variable-width bins with equal cross section are constructed automatically. Default value: None.

**histrange2**

[tuple of float or None, optional] Minimum and maximum value of the first axis of the histogram in the form  $(min, max)$ . Overflow bins are always added. If None, variable-width bins with equal cross section are constructed automatically. Default value: None.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**n\_events\_dynamic\_binning**

[int or None, optional] Number of events used to calculate the dynamic binning (if `histrange` is None). If None, all events are used. Note that these events are not shuffled, so if the events in the MadMiner file are sorted, using a value different from None can cause issues. Default value: None.

**Returns****fisher\_information**

[ndarray] Expected Fisher information in the histogram with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

**calculate\_fisher\_information\_nuisance\_constraints()**

Builds the Fisher information term representing the Gaussian constraints on the nuisance parameters

**calculate\_fisher\_information\_rate**(*theta*, *luminosity*, *cuts*=None, *efficiency\_functions*=None, *include\_nuisance\_parameters*=True)

Calculates the Fisher information in a measurement of the total cross section (without any kinematic information).

**Parameters****theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(theta)$  is evaluated.

**luminosity**

[float] Luminosity in  $\text{pb}^{-1}$ .

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

**Returns**

**fisher\_information**

[ndarray] Expected Fisher information in the total cross section with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

**full\_information**(*theta*, *model\_file*, *unweighted\_x\_sample\_file*=None, *luminosity*=300000.0, *include\_xsec\_info*=True, *mode*='score', *calculate\_covariance*=True, *batch\_size*=100000, *test\_split*=0.2)

Calculates the full Fisher information in realistic detector-level observations, estimated with neural networks. In addition to the MadMiner file, this requires a trained SALLY or SALLINO estimator.

Nuisance parameter are taken into account automatically if the SALLY / SALLINO model was trained with them.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**model\_file**

[str] Filename of a trained local score regression model that was trained on samples from *theta* (see *madminer.ml.Estimator*).

**unweighted\_x\_sample\_file**

[str or None] Filename of an unweighted x sample that is sampled according to *theta* and obeys the cuts (see *madminer.sampling.SampleAugmenter.extract\_samples\_train\_local()*). If None, the Fisher information is instead calculated on the full, weighted samples (the data in the MadMiner file). Default value: None.

**luminosity**

[float, optional] Luminosity in pb<sup>-1</sup>. Default value: 300000.

**include\_xsec\_info**

[bool, optional] Whether the rate information is included in the returned Fisher information. Default value: True.

**mode**

[{"score", "information"}, optional] How the ensemble uncertainty on the kinematic Fisher information is calculated. If mode is "information", the Fisher information for each estimator is calculated individually and only then are the sample mean and covariance calculated. If mode is "score", the sample mean is calculated for the score for each event. Default value: "score".

**calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: True.

**batch\_size**

[int, optional] Batch size. Default value: 100000.

**test\_split**

[float or None, optional] If `unweighted_x_sample_file` is None, this determines the fraction of weighted events used for evaluation. If None, all events are used (this will probably include events used during training!). Default value: 0.2.

**Returns**

**fisher\_information**

[ndarray or list of ndarray] Estimated expected full detector-level Fisher information matrix with shape  $(n\_parameters, n\_parameters)$ . If more then one value `ensemble_vote_expectation_weight` is given, this is a list with results for all entries in `ensemble_vote_expectation_weight`.

**fisher\_information\_uncertainty**

[ndarray or list of ndarray or None] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ . If more then one value `ensemble_vote_expectation_weight` is given, this is a list with results for all entries in `ensemble_vote_expectation_weight`.

**histo\_information**(*theta*, *luminosity*, *observable*, *bins*, *histrange*=None, *cuts*=None, *efficiency\_functions*=None, *n\_events\_dynamic\_binning*=None)

Calculates the Fisher information in the one-dimensional histogram of an (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observable.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**luminosity**

[float] Luminosity in  $\text{pb}^{-1}$ .

**observable**

[str] Expression for the observable to be histogrammed. The str will be parsed by Python's `eval()` function and can use the names of the observables in the MadMiner files.

**bins**

[int or ndarray] If int: number of bins in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries (excluding overflow bins).

**histrange**

[tuple of float or None, optional] Minimum and maximum value of the histogram in the form  $(min, max)$ . Overflow bins are always added. If None and bins is an int, variable-width bins with equal cross section are constructed automatically. Default value: None.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**n\_events\_dynamic\_binning**

[int or None, optional] Number of events used to calculate the dynamic binning (if *histrange* is None). If None, all events are used. Note that these events are not shuffled, so if the events in the MadMiner file are sorted, using a value different from None can cause issues. Default value: None.

**Returns**

**fisher\_information**

[ndarray] Expected Fisher information in the histogram with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

**histo\_information\_2d**(*theta*, *luminosity*, *observable1*, *bins1*, *observable2*, *bins2*, *histrange1*=None, *histrange2*=None, *cuts*=None, *efficiency\_functions*=None, *n\_events\_dynamic\_binning*=None)

Calculates the Fisher information in a two-dimensional histogram of two (parton-level or detector-level, depending on how the observations in the MadMiner file were calculated) observables.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(theta)$  is evaluated.

**luminosity**

[float] Luminosity in  $\text{pb}^{-1}$ .

**observable1**

[str] Expression for the first observable to be histogrammed. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**bins1**

[int or ndarray] If int: number of bins along the first axis in the histogram in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries along the first axis in the histogram (excluding overflow bins).

**observable2**

[str] Expression for the first observable to be histogrammed. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**bins2**

[int or ndarray] If int: number of bins along the second axis in the histogram in the histogram, excluding overflow bins. Otherwise, defines the bin boundaries along the second axis in the histogram (excluding overflow bins).

**histrange1**

[tuple of float or None, optional] Minimum and maximum value of the first axis of the histogram in the form  $(min, max)$ . Overflow bins are always added. If None, variable-width bins with equal cross section are constructed automatically. Default value: None.

**histrange2**

[tuple of float or None, optional] Minimum and maximum value of the first axis of the histogram in the form  $(min, max)$ . Overflow bins are always added. If None, variable-width bins with equal cross section are constructed automatically. Default value: None.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**n\_events\_dynamic\_binning**

[int or None, optional] Number of events used to calculate the dynamic binning (if *histrange* is None). If None, all events are used. Note that these events are not shuffled, so if the events in the MadMiner file are sorted, using a value different from None can cause issues. Default value: None.

**Returns**

**fisher\_information**

[ndarray] Expected Fisher information in the histogram with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

**histogram\_of\_fisher\_information**(*theta*, *observable*, *nbins*, *histrange*, *model\_file*=None, *luminosity*=300000.0, *cuts*=None, *efficiency\_functions*=None, *batch\_size*=100000, *test\_split*=0.2)

Calculates the full and rate-only Fisher information in slices of one observable. For the full information, it will return the truth-level information if *model\_file* is None, and otherwise the detector-level information based on the SALLY-type score estimator saved in *model\_file*.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**observable**

[str] Expression for the observable to be sliced. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**nbins**

[int] Number of bins in the slicing, excluding overflow bins.

**histrange**

[tuple of float] Minimum and maximum value of the slicing in the form  $(min, max)$ . Overflow bins are always added.

**model\_file**

[str or None, optional] If None, the truth-level Fisher information is calculated. If str, filename of a trained local score regression model that was trained on samples from *theta* (see *madminer.ml.Estimator*). Default value: None.

**luminosity**

[float, optional] Luminosity in  $\text{pb}^{-1}$ . Default value: 300000.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**batch\_size**

[int, optional] If model\_file is not None: Batch size. Default value: 100000.

**test\_split**

[float or None, optional] If model\_file is not None: If unweighted\_x\_sample\_file is None, this determines the fraction of weighted events used for evaluation. If None, all events are used (this will probably include events used during training!). Default value: 0.2.

**Returns**

**bin\_boundaries**

[ndarray] Observable slice boundaries.

**sigma\_bins**

[ndarray] Cross section in pb in each of the slices.

**fisher\_infos\_rate**

[ndarray] Expected rate-only Fisher information for each slice. Has shape  $(n\_slices, n\_parameters, n\_parameters)$ .

**fisher\_infos\_full**

[ndarray] Expected full Fisher information for each slice. Has shape  $(n\_slices, n\_parameters, n\_parameters)$ .

**histogram\_of\_information**(*theta*, *observable*, *nbins*, *histrange*, *model\_file*=None, *luminosity*=300000.0, *cuts*=None, *efficiency\_functions*=None, *batch\_size*=100000, *test\_split*=0.2)

Calculates the full and rate-only Fisher information in slices of one observable. For the full information, it will return the truth-level information if model\_file is None, and otherwise the detector-level information based on the SALLY-type score estimator saved in model\_file.

**Parameters**

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**observable**

[str] Expression for the observable to be sliced. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**nbins**

[int] Number of bins in the slicing, excluding overflow bins.

**histrange**

[tuple of float] Minimum and maximum value of the slicing in the form  $(min, max)$ . Overflow bins are always added.

**model\_file**

[str or None, optional] If None, the truth-level Fisher information is calculated. If str, filename of a trained local score regression model that was trained on samples from *theta* (see *madminer.ml.Estimator*). Default value: None.

**luminosity**

[float, optional] Luminosity in  $pb^{-1}$ . Default value: 300000.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**batch\_size**

[int, optional] If model\_file is not None: Batch size. Default value: 100000.

**test\_split**

[float or None, optional] If model\_file is not None: If unweighted\_x\_sample\_file is None, this determines the fraction of weighted events used for evaluation. If None, all events are used (this will probably include events used during training!). Default value: 0.2.

**Returns****bin\_boundaries**

[ndarray] Observable slice boundaries.

**sigma\_bins**

[ndarray] Cross section in pb in each of the slices.

**fisher\_infos\_rate**

[ndarray] Expected rate-only Fisher information for each slice. Has shape  $(n\_slices, n\_parameters, n\_parameters)$ .

**fisher\_infos\_full**

[ndarray] Expected full Fisher information for each slice. Has shape  $(n\_slices, n\_parameters, n\_parameters)$ .

**histogram\_of\_sigma\_dsigma**(*theta*, *observable*, *nbins*, *histrange*, *cuts*=None, *efficiency\_functions*=None)

Fills events into histograms and calculates the cross section and first derivative for each bin

**Parameters****theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**observable**

[str] Expression for the observable to be sliced. The str will be parsed by Python's *eval()* function and can use the names of the observables in the MadMiner files.

**nbins**

[int] Number of bins in the slicing, excluding overflow bins.

**histrange**

[tuple of float] Minimum and maximum value of the slicing in the form  $(min, max)$ . Overflow bins are always added.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**Returns****bin\_boundaries**

[ndarray] Observable slice boundaries.

**sigma\_bins**

[ndarray] Cross section in pb in each of the slices.

**dsigma\_bins**

[ndarray] Cross section in pb in each of the slices.

**nuisance\_constraint\_information()**

Builds the Fisher information term representing the Gaussian constraints on the nuisance parameters

**rate\_information**(*theta*, *luminosity*, *cuts*=None, *efficiency\_functions*=None, *include\_nuisance\_parameters*=True)

Calculates the Fisher information in a measurement of the total cross section (without any kinematic information).

#### Parameters

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**luminosity**

[float] Luminosity in pb<sup>-1</sup>.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.

**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

#### Returns

**fisher\_information**

[ndarray] Expected Fisher information in the total cross section with shape (*n\_parameters*, *n\_parameters*).

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape (*n\_parameters*, *n\_parameters*, *n\_parameters*, *n\_parameters*), calculated with plain Gaussian error propagation.

**truth\_information**(*theta*, *luminosity*=300000.0, *cuts*=None, *efficiency\_functions*=None, *include\_nuisance\_parameters*=True)

Calculates the full Fisher information at parton / truth level. This is the information in an idealized measurement where all parton-level particles with their charges, flavours, and four-momenta can be accessed with perfect accuracy, i.e. the latent variables  $z_{parton}$  can be measured directly.

#### Parameters

**theta**

[ndarray] Parameter point *theta* at which the Fisher information matrix  $I_{ij}(\theta)$  is evaluated.

**luminosity**

[float] Luminosity in pb<sup>-1</sup>.

**cuts**

[None or list of str, optional] Cuts. Each entry is a parseable Python expression that returns a bool (True if the event should pass a cut, False otherwise). Default value: None.



**efficiency\_functions**

[list of str or None] Efficiencies. Each entry is a parseable Python expression that returns a float for the efficiency of one component. Default value: None.

**include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

**Returns****fisher\_information**

[ndarray] Expected full truth-level Fisher information matrix with shape  $(n\_parameters, n\_parameters)$ .

**fisher\_information\_uncertainty**

[ndarray] Covariance matrix of the Fisher information matrix with shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ , calculated with plain Gaussian error propagation.

## 9.4 madminer.fisherinformation.manipulate module

```
madminer.fisherinformation.manipulate.profile_information(fisher_information,
                                                         remaining_components,
                                                         covariance=None,
                                                         error_propagation_n_ensemble=1000,
                                                         error_propagation_factor=0.001)
```

Calculates the profiled Fisher information matrix as defined in Appendix A.4 of arXiv:1612.05261.

**Parameters****fisher\_information**

[ndarray] Original  $n \times n$  Fisher information.

**remaining\_components**

[list of int] List with  $m$  entries, each an int with  $0 \leq \text{remaining\_components}[i] < n$ . Denotes which parameters are kept, and their new order. All other parameters are profiled out.

**covariance**

[ndarray or None, optional] The covariance matrix of the original Fisher information with shape  $(n, n, n, n)$ . If None, the error on the profiled information is not calculated. Default value: None.

**error\_propagation\_n\_ensemble**

[int, optional] If covariance is not None, this sets the number of Fisher information matrices drawn from a normal distribution for the Monte-Carlo error propagation. Default value: 1000.

**error\_propagation\_factor**

[float, optional] If covariance is not None, this factor multiplies the covariance of the distribution of Fisher information matrices. Smaller factors can avoid problems with ill-behaved Fisher information matrices. Default value:  $1.e-3$ .

**Returns****profiled\_fisher\_information**

[ndarray] Profiled  $m \times m$  Fisher information, where the  $i$ -th row or column corresponds to the *remaining\_components*[ $i$ ]-th row or column of *fisher\_information*.

**profiled\_fisher\_information\_covariance**

[ndarray] Covariance matrix of the profiled Fisher information matrix with shape (m, m, m, m).

```
madminer.fisherinformation.manipulate.project_information(fisher_information,
                                                         remaining_components,
                                                         covariance=None)
```

Calculates projections of a Fisher information matrix, that is, “deletes” the rows and columns corresponding to some parameters not of interest.

**Parameters**

**fisher\_information**

[ndarray] Original n x n Fisher information.

**remaining\_components**

[list of int] List with m entries, each an int with  $0 \leq \text{remaining\_components}[i] < n$ . Denotes which parameters are kept, and their new order. All other parameters are projected out.

**covariance**

[ndarray or None, optional] The covariance matrix of the original Fisher information with shape (n, n, n, n). If None, the error on the profiled information is not calculated. Default value: None.

**Returns**

**projected\_fisher\_information**

[ndarray] Projected m x m Fisher information, where the *i*-th row or column corresponds to the *remaining\_components*[*i*]-th row or column of *fisher\_information*.

**profiled\_fisher\_information\_covariance**

[ndarray] Covariance matrix of the projected Fisher information matrix with shape (m, m, m, m). Only returned if covariance is not None.

## 9.5 Module contents

## MADMINER.LHE PACKAGE

### 10.1 Submodules

### 10.2 `madminer.lhe.lhe_reader` module

**class** `madminer.lhe.lhe_reader.LHEReader(filename)`

Bases: `object`

Detector simulation with smearing functions and simple calculation of observables.

After setting up the parameter space and benchmarks and running MadGraph and Pythia, all of which is organized in the `madminer.core.MadMiner` class, the next steps are the simulation of detector effects and the calculation of observables. Different tools can be used for these tasks, please feel free to implement the detector simulation and analysis routine of your choice.

This class provides a simple implementation in which detector effects are modeled with smearing functions. Its workflow consists of the following steps:

- Initializing the class with the filename of a MadMiner HDF5 file (the output of `madminer.core.MadMiner.save()`)
- Adding one or multiple event samples produced by MadGraph and Pythia in `LHEProcessor.add_sample()`.
- Running Delphes on the samples that require it through `LHEProcessor.run_delphes()`.
- Optionally, smearing functions for all visible particles can be defined with `LHEProcessor.set_smearing()`.
- Defining observables through `LHEProcessor.add_observable()` or `LHEProcessor.add_observable_from_function()`. A simple set of default observables is provided in `LHEProcessor.add_default_observables()`
- Optionally, cuts can be set with `LHEProcessor.add_cut()`
- Optionally, efficiencies can be set with `LHEProcessor.add_efficiency()`
- Calculating the observables from the Delphes ROOT files with `LHEProcessor.analyse_delphes_samples()`
- Saving the results with `LHEProcessor.save()`

Please see the tutorial for a detailed walk-through.

#### Parameters

##### **filename**

[str or None, optional] Path to MadMiner file (the output of `madminer.core.MadMiner.save()`). Default value: None.

## Methods

<code>add_cut(definition[, required])</code>	Adds a cut as a string that can be parsed by Python's <code>eval()</code> function and returns a bool.
<code>add_default_observables([n_leptons_max, ...])</code>	Adds a set of simple standard observables: the four-momenta (parameterized as E, pT, eta, phi) of the hardest visible particles, and the missing transverse energy.
<code>add_efficiency(definition[, default])</code>	Adds an efficiency as a string that can be parsed by Python's <code>eval()</code> function and returns a bool.
<code>add_observable(name, definition[, required, ...])</code>	Adds an observable as a string that can be parsed by Python's <code>eval()</code> function.
<code>add_observable_from_function(name, fn[, ...])</code>	Adds an observable defined through a function.
<code>add_sample(lhe_filename, sampled_from_benchmark)</code>	Adds an LHE sample of simulated events.
<code>analyse_samples([reference_benchmark, ...])</code>	Main function that parses the LHE samples, applies detector effects, checks cuts, evaluate efficiencies, and extracts the observables and weights.
<code>reset_cuts()</code>	Resets all cuts.
<code>reset_efficiencies()</code>	Resets all efficiencies.
<code>reset_observables()</code>	Resets all observables.
<code>save(filename_out[, shuffle])</code>	Saves the observable definitions, observable values, and event weights in a MadMiner file.
<code>set_met_noise([abs_, rel])</code>	Sets up additional noise in the MET variable from shower and detector effects.
<code>set_smearing([pdgids, ...])</code>	Sets up the smearing of measured momenta from shower and detector effects.

**add\_cut** (*definition*, *required=False*)

Adds a cut as a string that can be parsed by Python's `eval()` function and returns a bool.

### Parameters

#### definition

[str] An expression that can be parsed by Python's `eval()` function and returns a bool: True for the event to pass this cut, False for it to be rejected. In the definition, all visible particles can be used: *e*, *mu*, *j*, *a*, and *l* provide lists of electrons, muons, jets, photons, and leptons (electrons and muons combined), in each case sorted by descending transverse momentum. *met* provides a missing ET object. *visible* and *all* provide access to the sum of all visible particles and the sum of all visible particles plus MET, respectively. In addition, *MadMinerParticle* have properties *charge* and *pdg\_id*, which return the charge in units of elementary charges (i.e. an electron has *e[0].charge = -1.*), and the PDG particle ID. For instance, "*len(e) >= 2*" requires at least two electrons passing the cuts, while "*mu[0].charge > 0.*" specifies that the hardest muon is positively charged.

#### required

[bool, optional] Whether the cut is passed if the observable cannot be parsed. Default value: False.

### Returns

None

**add\_default\_observables** (*n\_leptons\_max=2*, *n\_photons\_max=2*, *n\_jets\_max=2*, *include\_met=True*, *include\_visible\_sum=True*, *include\_numbers=True*, *include\_charge=True*)

Adds a set of simple standard observables: the four-momenta (parameterized as E, pT, eta, phi) of the hardest visible particles, and the missing transverse energy.

#### Parameters

##### **n\_leptons\_max**

[int, optional] Number of hardest leptons for which the four-momenta are saved. Default value: 2.

##### **n\_photons\_max**

[int, optional] Number of hardest photons for which the four-momenta are saved. Default value: 2.

##### **n\_jets\_max**

[int, optional] Number of hardest jets for which the four-momenta are saved. Default value: 2.

##### **include\_met**

[bool, optional] Whether the missing energy observables are stored. Default value: True.

##### **include\_visible\_sum**

[bool, optional] Whether observables characterizing the sum of all particles are stored. Default value: True.

##### **include\_numbers**

[bool, optional] Whether the number of leptons, photons, and jets is saved as observable. Default value: True.

##### **include\_charge**

[bool, optional] Whether the lepton charge is saved as observable. Default value: True.

#### Returns

None

**add\_efficiency**(*definition*, *default=1.0*)

Adds an efficiency as a string that can be parsed by Python's *eval()* function and returns a bool.

#### Parameters

##### **definition**

[str] An expression that can be parsed by Python's *eval()* function and returns a floating number which reweights the event weights. In the definition, all visible particles can be used: *e*, *mu*, *j*, *a*, and *l* provide lists of electrons, muons, jets, photons, and leptons (electrons and muons combined), in each case sorted by descending transverse momentum. *met* provides a missing ET object. *visible* and *all* provide access to the sum of all visible particles and the sum of all visible particles plus MET, respectively. In addition, *MadMiner-Particle* have properties *charge* and *pdg\_id*, which return the charge in units of elementary charges (i.e. an electron has *e[0].charge = -1.*), and the PDG particle ID.

##### **default**

[float, optional] Value if the efficiency function cannot be parsed. Default value: 1.

#### Returns

None

**add\_observable**(*name*, *definition*, *required=False*, *default=None*)

Adds an observable as a string that can be parsed by Python's *eval()* function.

#### Parameters

**name**

[str] Name of the observable. Since this name will be used in *eval()* calls for cuts, this should not contain spaces or special characters.

**definition**

[str] An expression that can be parsed by Python's *eval()* function. As objects, all particles can be used: *e*, *mu*, *tau*, *j*, *a*, *l*, *v* provide lists of electrons, muons, taus, jets, photons, leptons (electrons and muons combined), and neutrinos, in each case sorted by descending transverse momentum. *met* provides a missing ET object. *p* gives all particles in the same order as in the LHE file (i.e. in the same order as defined in the MadGraph process card). In addition, *MadMinerParticle* have properties *charge* and *pdg\_id*, which return the charge in units of elementary charges (i.e. an electron has *e[0].charge = -1.*), and the PDG particle ID. For instance, "*abs(j[0].phi - j[1].phi)*" defines the azimuthal angle between the two hardest jets.

**required**

[bool, optional] Whether the observable is required. If True, an event will only be retained if this observable is successfully parsed. For instance, any observable involving "*j[1]*" will only be parsed if there are at least two jets passing the acceptance cuts. Default value: False.

**default**

[float or None, optional] If *required=False*, this is the placeholder value for observables that cannot be parsed. None is replaced with *np.nan*. Default value: None.

**Returns**

None

**add\_observable\_from\_function**(*name*, *fn*, *required=False*, *default=None*)

Adds an observable defined through a function.

**Parameters**

**name**

[str] Name of the observable. Since this name will be used in *eval()* calls for cuts, this should not contain spaces or special characters.

**fn**

[function] A function with signature *observable(particles, leptons, photons, jets, met)* where all arguments are lists of *MadMinerParticle* instances and a float is returned. *particles* are the truth-level particles, ordered in the same way as in the LHE file, and no smearing is applied. *leptons*, *photons*, *jets*, and *met* have smearing applied. The function should raise a *RuntimeError* to signal that it is not defined.

**required**

[bool, optional] Whether the observable is required. If True, an event will only be retained if this observable is successfully parsed. For instance, any observable involving "*j[1]*" will only be parsed if there are at least two jets passing the acceptance cuts. Default value: False.

**default**

[float or None, optional] If *required=False*, this is the placeholder value for observables that cannot be parsed. None is replaced with *np.nan*. Default value: None.

**Returns**

None

**add\_sample**(*lhe\_filename*, *sampled\_from\_benchmark*, *is\_background=False*, *k\_factor=1.0*, *systematics=None*)

Adds an LHE sample of simulated events.

#### Parameters

##### **lhe\_filename**

[str] Path to the LHE event file (with extension `‘.lhe’` or `‘.lhe.gz’`).

##### **sampled\_from\_benchmark**

[str] Name of the benchmark that was used for sampling in this event file (the keyword `sample_benchmark` of `madminer.core.MadMiner.run()`).

##### **is\_background**

[bool, optional] Whether the sample is a background sample (i.e. without benchmark reweighting).

##### **k\_factor**

[float, optional] Multiplies the cross sections found in the sample. Default value: 1.

##### **systematics**

[None or list of str, optional] List of systematics associated with this sample. Default value: None.

#### Returns

None

**analyse\_samples**(*reference\_benchmark=None*, *parse\_events\_as\_xml=True*)

Main function that parses the LHE samples, applies detector effects, checks cuts, evaluate efficiencies, and extracts the observables and weights.

#### Parameters

##### **reference\_benchmark**

[str or None, optional] The weights at the nuisance benchmarks will be rescaled to some reference theta benchmark:  $dsigma(x|theta\_sampling(x),nu) \rightarrow dsigma(x|theta\_ref,nu) = dsigma(x|theta\_sampling(x),nu) * dsigma(x|theta\_ref,0) / dsigma(x|theta\_sampling(x),0)$ . This sets the name of the reference benchmark. If None, the first one will be used. Default value: None.

##### **parse\_events\_as\_xml**

[bool, optional] Decides whether the LHE events are parsed with an XML parser (more robust, but slower) or a text parser (less robust, faster). Default value: True.

#### Returns

None

**reset\_cuts()**

Resets all cuts.

**reset\_efficiencies()**

Resets all efficiencies.

**reset\_observables()**

Resets all observables.

**save**(*filename\_out*, *shuffle=True*)

Saves the observable definitions, observable values, and event weights in a MadMiner file. The parameter, benchmark, and morphing setup is copied from the file provided during initialization. Nuisance benchmarks found in the LHE file are added.

**Parameters****filename\_out**

[str] Path to where the results should be saved.

**shuffle**

[bool, optional] If True, events are shuffled before being saved. That's important when there are multiple distinct samples (e.g. signal and background). Default value: True.

**Returns**

None

**set\_met\_noise**(*abs=0.0, rel=0.0*)

Sets up additional noise in the MET variable from shower and detector effects.

By default, the MET is calculated based on all reconstructed visible particles, including the effect of the smearing of these particles (set with *set\_smearing()*). But often the MET is in fact more affected by additional soft activity than by mismeasurements of the hard particles. This function adds a Gaussian random to each of the x and y components of the MET vector. The Gaussian has mean 0 and standard deviation  $abs + rel * HT$ , where  $HT$  is the scalar sum of the pT of all particles in the process. Everything is given in GeV.

**Parameters****abs\_**

[float, optional] Absolute contribution to MET noise. Default value: 0.

**rel**

[float, optional] Relative (to HT) contribution to MET noise. Default value: 0.

**Returns**

None

**set\_smearing**(*pdgids=None, energy\_resolution\_abs=0.0, energy\_resolution\_rel=0.0, pt\_resolution\_abs=0.0, pt\_resolution\_rel=0.0, eta\_resolution\_abs=0.0, eta\_resolution\_rel=0.0, phi\_resolution\_abs=0.0, phi\_resolution\_rel=0.0*)

Sets up the smearing of measured momenta from shower and detector effects.

This function can be called with *pdgids=None*, in which case the settings are used for all visible particles, or with *pdgids* set to a list of PDG ids representing particles, for instance [11, -11] for electrons (and positrons).

For all particles of this type, and for the energy, pT, phi, and eta, the measurement error is drawn from a Gaussian with mean 0 and standard deviation given by  $(X\_resolution\_abs + X * X\_resolution\_rel)$ . Here  $X$  is the quantity (E, pT, phi, eta) of interest and  $X\_resolution\_abs$  and  $X\_resolution\_rel$  are the corresponding keywords. In the case of energy and pT, values smaller than 0 will lead to a re-drawing of the measurement error.

Instead of such numerical values, either the energy or pT resolution (but not both!) may be set to None. In this case, this quantity is calculated from the mass of the particle and all other smeared particles. For instance, when *pt\_resolution\_abs* is None or *pt\_resolution\_rel* is None, for the given particles the energy, phi, and eta are smeared (according to their respective resolutions). Then the transverse momentum is calculated from the on-shell condition  $p^2 = m^2$ , or  $pT = \sqrt{E^2 - m^2} / \cosh(eta)$ . When this does not have a solution, the pT is set to zero. On the other hand, when *energy\_resolution\_abs* is None or *energy\_resolution\_rel* is None, for the given particles the pT, phi, and eta are smeared, and then the energy is calculated as  $E = \sqrt{pT^2 * \cosh(eta)^2 + m^2}$ .

**Parameters****pdgids**

[None or list of int, optional] Defines the particles these smearing functions affect. If None,



all particles are affected. Note that if `set_smearing()` is called multiple times for a given particle, the earlier calls will be forgotten and only the last smearing function will take effect. Default value: `None`.

**energy\_resolution\_abs**

[float or `None`, optional] Absolute measurement uncertainty for the energy in GeV. `None` means that the energy is not smeared directly, but calculated from the on-shell condition. Default value: 0.

**energy\_resolution\_rel**

[float or `None`, optional] Relative measurement uncertainty for the energy. `None` means that the energy is not smeared directly, but calculated from the on-shell condition. Default value: 0.

**pt\_resolution\_abs**

[float or `None`, optional] Absolute measurement uncertainty for the pT in GeV. `None` means that the pT is not smeared directly, but calculated from the on-shell condition. Default value: 0.

**pt\_resolution\_rel**

[float or `None`, optional] Relative measurement uncertainty for the pT. `None` means that the pT is not smeared directly, but calculated from the on-shell condition. Default value: 0.

**eta\_resolution\_abs**

[float, optional] Absolute measurement uncertainty for eta. Default value: 0.

**eta\_resolution\_rel**

[float, optional] Relative measurement uncertainty for eta. Default value: 0.

**phi\_resolution\_abs**

[float, optional] Absolute measurement uncertainty for phi. Default value: 0.

**phi\_resolution\_rel**

[float, optional] Relative measurement uncertainty for phi. Default value: 0.

**Returns**

`None`

## 10.3 Module contents



## MADMINER.LIKELIHOOD PACKAGE

### 11.1 Submodules

### 11.2 `madminer.likelihood.base` module

**class** `madminer.likelihood.base.BaseLikelihood`(*filename*, *disable\_morphing=False*,  
*include\_nuisance\_parameters=True*)

Bases: *DataAnalyzer*

#### Methods

<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

<code>create_expected_negative_log_likelihood</code>	
<code>create_negative_log_likelihood</code>	

**abstract** `create_expected_negative_log_likelihood(*args, **kwargs)`

**abstract** `create_negative_log_likelihood(*args, **kwargs)`

### 11.3 `madminer.likelihood.histo` module

**class** `madminer.likelihood.histo.HistoLikelihood`(*filename*, *disable\_morphing=False*,  
*include\_nuisance\_parameters=True*)

Bases: *BaseLikelihood*

## Methods

<code>create_expected_negative_log_likelihood(...)</code>	Returns a function which calculates the expected negative log likelihood for a given parameter point, evaluated with test data sampled according to <code>theta_true</code> .
<code>create_negative_log_likelihood(x_observed[, ...])</code>	Returns a function which calculates the negative log likelihood for a given parameter point, evaluated with a dataset ( <code>x_observed</code> , <code>n_observed</code> , <code>x_observed_weights</code> ).
<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if <code>theta</code> is <code>None</code> ) or weights for a given <code>theta</code> .
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

**create\_expected\_negative\_log\_likelihood**(*theta\_true*, *nu\_true*, *observables*=None, *score\_components*=None, *include\_xsec*=True, *luminosity*=300000.0, *n\_asimov*=None, *mode*='sampled', *n\_histo\_toys*=100000, *model\_file*=None, *hist\_bins*=None, *thetas\_binning*=None, *test\_split*=None)

Returns a function which calculates the expected negative log likelihood for a given parameter point, evaluated with test data sampled according to `theta_true`.

### Parameters

#### **theta\_true**

[ndarray] Specifies the physical parameters according to which the test data is sampled.

#### **nu\_true**

[ndarray] Specifies the nuisance parameters according to which the test data is sampled.

#### **observables**

[list of str or None, optional] Kinematic variables used in the histograms. The names are the same as used for instance in *DelphesReader*.

#### **score\_components**

[None or list of int, optional] Defines the score components used. Default value: None.

#### **include\_xsec**

[bool, optional] Whether the Poisson likelihood representing the total number of events is included in the analysis. Default value: True.

#### **luminosity**

[float, optional] Integrated luminosity in pb<sup>-1</sup> assumed in the analysis. Default value: 300000.

#### **n\_asimov**

[int or None, optional] Size of the Asimov sample. If None, all weighted events in the MadMiner file are used. Default value: None.

#### **mode**

[{"weighted", "sampled"}, optional] If "sampled", for each evaluation of the likelihood function, a separate set of events are sampled and histogram is created to construct the likelihood function. If "weighted", first a set of weighted events is sampled which is then used to create histograms. Default value: "sampled"

**n\_histo\_toys**

[int or None, optional] Number of events drawn to construct the histograms used. If None and weighted\_histo is True, all events in the training fraction of the MadMiner file are used. If None and weighted\_histo is False, 100000 events are used. Default value: 100000.

**model\_file**

[str or None, optional] Filename of a saved neural network estimating the score. Required if score\_components is not None. Default value: None.

**hist\_bins**

[int or list of (int or ndarray) or None, optional] Defines the histogram binning. If int, gives the number of bins automatically chosen for each summary statistic. If list, each entry corresponds to one summary statistic (e.g. kinematic variable specified by hist\_vars); an int entry corresponds to the number of automatically chosen bins, an ndarray specifies the bin edges along this dimension explicitly. If None, the bins are chosen according to the defaults: for one summary statistic the default is 25 bins, for 2 it's 8 bins along each direction, for more it's 5 per dimension. Default value: None.

**thetas\_binning**

[list of ndarray or None] Specifies the parameter points used to determine the optimal binning. If none, theta\_true will be used. Default value : None

**test\_split**

**Returns**

**negative\_log\_likelihood**

[likelihood] Function that evaluates the negative log likelihood for a given parameter point

```
create_negative_log_likelihood(x_observed, observables=None, score_components=None,
                               n_observed=None, x_observed_weights=None, include_xsec=True,
                               luminosity=300000.0, mode='sampled', n_histo_toys=100000,
                               model_file=None, hist_bins=None, thetas_binning=None,
                               test_split=None)
```

Returns a function which calculates the negative log likelihood for a given parameter point, evaluated with a dataset (x\_observed, n\_observed, x\_observed\_weights).

**Parameters**

**x\_observed**

[list of ndarray] Set of event observables with shape (*n\_events*, *n\_observables*).

**observables**

[list of str or None, optional] Kinematic variables used in the histograms. The names are the same as used for instance in *DelphesReader*.

**score\_components**

[None or list of int, optional] Defines the score components used. Default value: None.

**n\_observed**

[int or None, optional] If int, number of observed events. If None, n\_observed is defined by the length of x\_observed. Default: None.

**x\_observed\_weights**

[list of float or None, optional] List of event weights with shape (*n\_events*). If None, all events have equal weights. Default: None.

**include\_xsec**

[bool, optional] Whether the Poisson likelihood representing the total number of events is included in the analysis. Default value: True.

**luminosity**

[float, optional] Integrated luminosity in  $\text{pb}^{-1}$  assumed in the analysis. Default value: 300000.

**mode**

[{"weighted", "sampled", "histo"}, optional] If "sampled", for each evaluation of the likelihood function, a separate set of events are sampled and histogram is created to construct the likelihood function. If "weighted", first a set of weighted events is sampled which is then used to create histograms. Default value: "sampled"

**n\_histo\_toys**

[int or None, optional] Number of events drawn to construct the histograms used. If None and weighted\_histo is True, all events in the training fraction of the MadMiner file are used. If None and weighted\_histo is False, 100000 events are used. Default value: 100000.

**model\_file**

[str or None, optional] Filename of a saved neural network estimating the score. Required if score\_components is not None. Default value: None.

**hist\_bins**

[int or list of (int or ndarray) or None, optional] Defines the histogram binning. If int, gives the number of bins automatically chosen for each summary statistic. If list, each entry corresponds to one summary statistic (e.g. kinematic variable specified by hist\_vars); an int entry corresponds to the number of automatically chosen bins, an ndarray specifies the bin edges along this dimension explicitly. If None, the bins are chosen according to the defaults: for one summary statistic the default is 25 bins, for 2 it's 8 bins along each direction, for more it's 5 per dimension. Default value: None.

**thetas\_binning**

[list of ndarray or None] Specifies the parameter points used to determine the optimal binning. This is required if hist\_bins doesn't already fully specify the binning of the histogram. Default value : None

**test\_split**

**Returns**

**negative\_log\_likelihood**

[likelihood] Function that evaluates the negative log likelihood for a given parameter point

## 11.4 madminer.likelihood.manipulate module

`madminer.likelihood.manipulate.fix_params(negative_log_likelihood, theta, fixed_components=None)`

Function that reduces the dimensionality of a likelihood function by fixing some of the components.

**Parameters**

**negative\_log\_likelihood**

[likelihood] Function returned by Likelihood class (for example `NeuralLikelihood.create_expected_negative_log_likelihood()`) which takes an n-dimensional input parameter.

**theta**

[list of float] m-dimensional vector of coordinate which will be fixed.

**fixed\_components**

[list of int or None, optional.] m-dimensional vector of coordinate indices provided in theta.

*fixed\_components=[0,1]* will fix the 1st and 2nd component of the parameter point. If None, uses [0, ..., m-1].

## Returns

### **constrained\_nll\_negative\_log\_likelihood**

[likelihood] Constrained likelihood function which takes an n-m dimensional input parameter.

```
madminer.likelihood.manipulate.profile_log_likelihood(negative_log_likelihood,
                                                    remaining_components=None,
                                                    grid_ranges=None, grid_resolutions=25,
                                                    thetas_eval=None, theta_start=None,
                                                    dof=None, method='TNC')
```

Takes a likelihood function depending on N parameters, and evaluates for a set of M-dimensional parameter points (either grid or explicitly specified) while the remaining N-M parameters are profiled over.

## Parameters

### **negative\_log\_likelihood**

[likelihood] Function returned by Likelihood class (for example NeuralLikelihood.create\_expected\_negative\_log\_likelihood()).

### **remaining\_components**

[list of int or None , optional] List with M entries, each an int with  $0 \leq \text{remaining\_components}[i] < N$ . Denotes which parameters are kept, and their new order. All other parameters or projected out (set to zero). If None, all components are kept. Default: None

### **grid\_ranges**

[list of (tuple of float) or None, optional] Specifies the boundaries of the parameter grid on which the p-values are evaluated. It should be [(min, max), (min, max), ..., (min, max)], where the list goes over all parameters and min and max are float. If None, thetas\_eval has to be given. Default: None.

### **grid\_resolutions**

[int or list of int, optional] Resolution of the parameter space grid on which the p-values are evaluated. If int, the resolution is the same along every dimension of the hypercube. If list of int, the individual entries specify the number of points along each parameter individually. Doesn't have any effect if grid\_ranges is None. Default value: 25.

### **thetas\_eval**

[ndarray or None , optional] Manually specifies the parameter point at which the likelihood and p-values are evaluated. If None, grid\_ranges and resolution are used instead to construct a regular grid. Default value: None.

### **theta\_start**

[ndarray or None , optional] Manually specifies a parameter point which is the starting point for the minimization algorithm which find the maximum likelihood point. If None, theta\_start = 0 is used. Default is None.

### **dof**

[int or None, optional] If not None, sets the number of parameters for the calculation of the p-values. If None, the overall number of parameters is used. Default value: None.

### **method**

[{"TNC", "L-BFGS-B"} , optional] Minimization method used. Default value: "TNC"

## Returns

**parameter\_grid**

[ndarray] Parameter points at which the p-values are evaluated with shape  $(n\_grid\_points, n\_parameters)$ .

**p\_values**

[ndarray] Observed p-values for each parameter point on the grid, with shape  $(n\_grid\_points,)$ .

**mle**

[int] Index of the parameter point with the best fit (largest p-value / smallest -2 log likelihood ratio).

**log\_likelihood\_ratio**

[ndarray or None] log likelihood ratio based only on kinematics for each point of the grid, with shape  $(n\_grid\_points,)$ .

`madminer.likelihood.manipulate.project_log_likelihood(negative_log_likelihood, remaining_components=None, grid_ranges=None, grid_resolutions=25, dof=None, thetas_eval=None)`

Takes a likelihood function depending on N parameters, and evaluates for a set of M-dimensional parameter points (either grid or explicitly specified) while the remaining N-M parameters are set to zero.

**Parameters**

**negative\_log\_likelihood**

[likelihood] Function returned by Likelihood class (for example `NeuralLikelihood.create_expected_negative_log_likelihood()`).

**remaining\_components**

[list of int or None , optional] List with M entries, each an int with  $0 \leq \text{remaining\_components}[i] < N$ . Denotes which parameters are kept, and their new order. All other parameters or projected out (set to zero). If None, all components are kept. Default: None

**grid\_ranges**

[list of (tuple of float) or None, optional] Specifies the boundaries of the parameter grid on which the p-values are evaluated. It should be  $[(min, max), (min, max), \dots, (min, max)]$ , where the list goes over all parameters and *min* and *max* are float. If None, *thetas\_eval* has to be given. Default: None.

**grid\_resolutions**

[int or list of int, optional] Resolution of the parameter space grid on which the p-values are evaluated. If int, the resolution is the same along every dimension of the hypercube. If list of int, the individual entries specify the number of points along each parameter individually. Doesn't have any effect if *grid\_ranges* is None. Default value: 25.

**dof**

[int or None, optional] If not None, sets the number of parameters for the calculation of the p-values. If None, the overall number of parameters is used. Default value: None.

**thetas\_eval**

[ndarray or None , optional] Manually specifies the parameter point at which the likelihood and p-values are evaluated. If None, *grid\_ranges* and resolution are used instead to construct a regular grid. Default value: None.

**Returns**

**parameter\_grid**

[ndarray] Parameter points at which the p-values are evaluated with shape  $(n\_grid\_points, n\_parameters)$ .



**p\_values**

[ndarray] Observed p-values for each parameter point on the grid, with shape  $(n\_grid\_points,)$ .

**mle**

[int] Index of the parameter point with the best fit (largest p-value / smallest -2 log likelihood ratio).

**log\_likelihood\_ratio**

[ndarray or None] log likelihood ratio based only on kinematics for each point of the grid, with shape  $(n\_grid\_points,)$ .

## 11.5 madminer.likelihood.neural module

```
class madminer.likelihood.neural.NeuralLikelihood(filename, disable_morphing=False,
                                                    include_nuisance_parameters=True)
```

Bases: *BaseLikelihood*

### Methods

<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

<code>create_expected_negative_log_likelihood</code>	
<code>create_negative_log_likelihood</code>	

```
create_expected_negative_log_likelihood(model_file, theta_true, nu_true, include_xsec=True,
                                         luminosity=300000.0, n_asimov=None, mode='sampled',
                                         n_weighted=10000, xsec_mode='interpol')
```

```
create_negative_log_likelihood(model_file, x_observed, n_observed=None,
                                x_observed_weights=None, include_xsec=True,
                                luminosity=300000.0, mode='weighted', n_weighted=10000,
                                xsec_mode='interpol')
```

## 11.6 Module contents

## MADMINER.LIMITS PACKAGE

### 12.1 Submodules

### 12.2 `madminer.limits.asymptotic_limits` module

```
class madminer.limits.asymptotic_limits.AsymptoticLimits(filename=None,  
                                                         include_nuisance_parameters=False)
```

Bases: *DataAnalyzer*

Statistical inference based on asymptotic properties of the likelihood ratio as test statistics.

This class provides two high-level functions:

- *AsymptoticLimits.observed\_limits()* calculates p-values over a grid in parameter space for a given set of observed data.
- *AsymptoticLimits.expected\_limits()* calculates expected p-values over a grid in parameter space based on “Asimov data”, a large hypothetical data set drawn from a given parameter point. This method is typically used to define expected exclusion limits or significances.

Both functions support inference based on...

- histograms of kinematic observables,
- based on histograms of score vectors estimated with the *madminer.ml.ScoreEstimator* class (SALLY and SALLINO techniques),
- based on likelihood or likelihood ratio functions estimated with the *madminer.ml.LikelihoodEstimator* and *madminer.ml.ParameterizedRatioEstimator* classes (NDE, SCANDAL, CARL, RASCAL, ALICES, and so on).

Currently, this class requires a morphing setup. It does *not* yet support nuisance parameters.

#### Parameters

##### **filename**

[str] Path to MadMiner file (for instance the output of *madminer.delphe.DelphesProcessor.save()*).

##### **include\_nuisance\_parameters**

[bool, optional] If True, nuisance parameters are taken into account. Currently not implemented. Default value: False.

## Methods

<code>asymptotic_p_value(log_likelihood_ratio[, dof])</code>	Calculates the p-value corresponding to a given log likelihood ratio and number of degrees of freedom assuming the asymptotic approximation.
<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>expected_limits(mode, theta_true[, ...])</code>	Calculates expected p-values over a grid in parameter space.
<code>observed_limits(mode, x_observed[, ...])</code>	Calculates p-values over a grid in parameter space based on a given set of observed events.
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

### **asymptotic\_p\_value**(log\_likelihood\_ratio, dof=None)

Calculates the p-value corresponding to a given log likelihood ratio and number of degrees of freedom assuming the asymptotic approximation.

#### Parameters

##### **log\_likelihood\_ratio**

[ndarray] Log likelihood ratio (without the factor -2)

##### **dof**

[int or None, optional] Number of parameters / degrees of freedom. None means the overall number of parameters is used. Default value: None.

#### Returns

##### **p\_values**

[ndarray] p-values.

**expected\_limits**(mode, theta\_true, grid\_ranges=None, grid\_resolutions=25, include\_xsec=True, model\_file=None, hist\_vars=None, score\_components=None, hist\_bins=None, thetaref=None, luminosity=300000.0, weighted\_histo=True, n\_histo\_toys=100000, histo\_theta\_batchsize=1000, dof=None, test\_split=0.2, return\_histos=True, return\_asimov=False, fix\_adaptive\_binning='auto-grid', sample\_only\_from\_closest\_benchmark=True, postprocessing=None, n\_asimov=None, n\_binning\_toys=100000, thetas\_eval=None)

Calculates expected p-values over a grid in parameter space.

*theta\_true* specifies which parameter point is assumed to be true. Based on this parameter point, the function generates a large artificial “Asimov data set”. p-values are then calculated with frequentist hypothesis tests using the likelihood ratio as test statistic. The asymptotic approximation is used, see <https://arxiv.org/abs/1007.1727>.

Depending on the keyword *mode*, the likelihood ratio is calculated with one of several different methods:

- With *mode*=“rate”, MadMiner only calculates the Poisson likelihood of the total number of events.
- With *mode*=“histo”, the kinematic likelihood is estimated with histograms of a small number of observables given by the keyword *hist\_vars*. *hist\_bins* determines the binning of the histograms. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.

- With *mode*="ml", the likelihood ratio is estimated with a parameterized neural network. *model\_file* has to point to the filename of a saved *LikelihoodEstimator* or *ParameterizedRatioEstimator* instance or a corresponding *Ensemble* (i.e. be the same filename used when calling *estimator.save()*). *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="sally", the likelihood ratio is estimated with histograms of the components of the estimated score vector. *model\_file* has to point to the filename of a saved *ScoreEstimator* instance. With *score\_components*, the histogram can be restricted to some components of the score. *hist\_bins* defines the binning of the histograms. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="adaptive-sally", the likelihood ratio is estimated with histograms of the components of the estimated score vector. The approach is essentially the same as for "sally", but the histogram binning is optimized for every parameter point by adding a new  $h = \text{score} * (\text{theta} - \text{thetaref})$  dimension to the histogram. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="sallino", the likelihood ratio is estimated with one-dimensional histograms of the scalar variable  $h = \text{score} * (\text{theta} - \text{thetaref})$  for each point *theta* along the parameter grid. *model\_file* has to point to the filename of a saved *ScoreEstimator* instance. *hist\_bins* defines the binning of the histogram. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.

MadMiner calculates one p-value for every parameter point on an evenly spaced grid specified by *grid\_ranges* and *grid\_resolutions*. For instance, in a three-dimensional parameter space, *grid\_ranges*=[(-1., 1.), (-2., 2.), (-3., 3.)] and *grid\_resolutions*=[10,10,10] will start the calculation along  $10^3$  parameter points in a cube with edges (-1, 1) in the first parameter and so on.

### Parameters

#### mode

[{"rate", "histo", "ml", "sally", "sallino", "adaptive-sally"}] Defines how the likelihood ratio test statistic is calculated. See above.

#### theta\_true

[ndarray] Parameter point assumed to be true to calculate the Asimov data.

#### grid\_ranges

[list of (tuple of float) or None, optional] Specifies the boundaries of the parameter grid on which the p-values are evaluated. It should be [(*min*, *max*), (*min*, *max*), ..., (*min*, *max*)], where the list goes over all parameters and *min* and *max* are float. If None, *thetas\_eval* has to be given. Default: None.

#### grid\_resolutions

[int or list of int, optional] Resolution of the parameter space grid on which the p-values are evaluated. If int, the resolution is the same along every dimension of the hypercube. If list of int, the individual entries specify the number of points along each parameter individually. Default value: 25.

#### include\_xsec

[bool, optional] Whether the Poisson likelihood representing the total number of events is included in the analysis. Default value: True.

#### model\_file

[str or None, optional] Filename of a saved neural network estimating the likelihood, likelihood ratio, or score. Required if mode is anything except "rate" or "histo". Default value: None.

#### hist\_vars

[list of str or None, optional] Kinematic variables used in the histograms when mode is

“histo”. The names are the same as used for instance in *DelphesReader*. Default value: None.

#### **score\_components**

[None or list of int, optional] Defines the score components used when mode is “sally” or “adaptive-sally”. Default value: None.

#### **hist\_bins**

[int or list of (int or ndarray) or None, optional] Defines the histogram binning when mode is “histo”, “sally”, “adaptive-sally”, or “sallino”. If int, gives the number of bins automatically chosen for each summary statistic. If list, each entry corresponds to one summary statistic (e.g. kinematic variable specified by `hist_vars` or estimated score component); an int entry corresponds to the number of automatically chosen bins, an ndarray specifies the bin edges along this dimension explicitly. If None, the bins are chosen according to the defaults: for one summary statistic the default is 25 bins, for 2 it’s 8 bins along each direction, for more it’s 5 per dimension. Default value: None.

#### **thetaref**

[ndarray or None, optional] Defines the reference parameter point at which the score is evaluated for mode “sallino” or “adaptive-sally”. If None, the origin in parameter space, [0., 0., ..., 0.], is used. Default value: None.

#### **luminosity**

[float, optional] Integrated luminosity in  $\text{pb}^{-1}$  assumed in the analysis. Default value: 300000.

#### **weighted\_histo**

[bool, optional] If True, the histograms used for the modes “histo”, “sally”, “sallino”, and “adaptive-sally” use one set of weighted events to construct the histograms at every point along the parameter grid, only with different weights for each parameter point on the grid. If False, independent unweighted event samples are drawn for each parameter point on the grid. Default value: True.

#### **n\_histo\_toys**

[int or None, optional] Number of events drawn to construct the histograms used for the modes “histo”, “sally”, “sallino”, and “adaptive-sally”. If None and `weighted_histo` is True, all events in the training fraction of the MadMiner file are used. If None and `weighted_histo` is False, 100000 events are used. Default value: 100000.

#### **histo\_theta\_batchsize**

[int or None, optional] Number of histograms constructed in parallel for the modes “histo”, “sally”, “sallino”, and “adaptive-sally” and if `weighted_histo` is True. A larger number speeds up the calculation, but requires more memory. Default value: 1000.

#### **dof**

[int or None, optional] If not None, sets the number of parameters for the calculation of the p-values. If None, the overall number of parameters is used. Default value: None.

#### **test\_split**

[float, optional] Fraction of weighted events in the MadMiner file reserved for evaluation. Default value: 0.2.

#### **return\_histos**

[bool, optional] If True and if mode is “histo”, “sally”, “adaptive-sally”, or “sallino”, the function returns histogram objects for each point along the grid.

#### **fix\_adaptive\_binning**

[[False, “center”, “grid”, “auto-grid”, “auto-center”], optional] If not False and if mode is “histo”, “sally”, “adaptive-sally”, or “sallino”, the automatic histogram binning is the same

for every point along the parameter grid. For “center”, the central point in the parameter grid is used to determine the binning, for “grid” all points in the parameter grid are combined for this. For “auto-grid” or “auto-center”, this option is turned on if mode is “histo” or “sally”, but not for “adaptive-sally” or “sallino”. Default value: “auto-grid”.

**sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only events originally generated from the closest benchmarks are used when generating the Asimov data (and, if weighted\_histo is False, the histogram data). Default value: True.

**return\_asimov**

[bool, optional] Whether the values of the summary statistics in the Asimov (“expected observed”) data set are returned. Default value: False.

**postprocessing**

[None or function, optional] If not None, points to a function that processes the summary statistics before being fed into histograms. Default value: None.

**n\_binning\_toys**

[int or None, optional] Number of toy events used to determine the binning of adaptive histograms. Default value: 100000.

**n\_asimov**

[int or None, optional] Size of the Asimov sample. If None, all weighted events in the MadMiner file are used. Default value: None.

**thetas\_eval**

[ndarray or None] Manually specifies the parameter point at which the likelihood and p-values are evaluated. If None, grid\_ranges and resolution are used instead to construct a regular grid. Default value: None.

**Returns**

**parameter\_grid**

[ndarray] Parameter points at which the p-values are evaluated with shape (*n\_grid\_points*, *n\_parameters*).

**p\_values**

[ndarray] Observed p-values for each parameter point on the grid, with shape (*n\_grid\_points*,).

**mle**

[int] Index of the parameter point with the best fit (largest p-value / smallest -2 log likelihood ratio).

**log\_likelihood\_ratio\_kin**

[ndarray or None] log likelihood ratio based only on kinematics for each point of the grid, with shape (*n\_grid\_points*,).

**log\_likelihood\_rate**

[ndarray or None] log likelihood based only on the total rate for each point of the grid, with shape (*n\_grid\_points*,).

**histos**

[None or list of Histogram] None if return\_histos is False. Otherwise a list of histogram objects for each point on the grid. This can be useful for debugging or for plotting the histograms.

```
observed_limits(mode, x_observed, grid_ranges=None, grid_resolutions=25, include_xsec=True,
                  model_file=None, hist_vars=None, score_components=None, hist_bins=None,
                  thetaref=None, luminosity=300000.0, weighted_histo=True, n_histo_toys=100000,
                  histo_theta_batchsize=1000, n_observed=None, dof=None, test_split=0.2,
                  return_histos=True, return_observed=False, fix_adaptive_binning='auto-grid',
                  postprocessing=None, n_binning_toys=100000, thetas_eval=None)
```

Calculates p-values over a grid in parameter space based on a given set of observed events.

*x\_observed* specifies the observed data as an array of observables, using the same observables and their order as used throughout the MadMiner workflow.

The p-values with frequentist hypothesis tests using the likelihood ratio as test statistic. The asymptotic approximation is used, see <https://arxiv.org/abs/1007.1727>.

Depending on the keyword *mode*, the likelihood ratio is calculated with one of several different methods:

- With *mode*="rate", MadMiner only calculates the Poisson likelihood of the total number of events.
- With *mode*="histo", the kinematic likelihood is estimated with histograms of a small number of observables given by the keyword *hist\_vars*. *hist\_bins* determines the binning of the histograms. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="ml", the likelihood ratio is estimated with a parameterized neural network. *model\_file* has to point to the filename of a saved *LikelihoodEstimator* or *ParameterizedRatioEstimator* instance or a corresponding *Ensemble* (i.e. be the same filename used when calling *estimator.save()*). *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="sally", the likelihood ratio is estimated with histograms of the components of the estimated score vector. *model\_file* has to point to the filename of a saved *ScoreEstimator* instance. With *score\_components*, the histogram can be restricted to some components of the score. *hist\_bins* defines the binning of the histograms. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="adaptive-sally", the likelihood ratio is estimated with histograms of the components of the estimated score vector. The approach is essentially the same as for "sally", but the histogram binning is optimized for every parameter point by adding a new  $h = \text{score} * (\text{theta} - \text{thetaref})$  dimension to the histogram. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.
- With *mode*="sallino", the likelihood ratio is estimated with one-dimensional histograms of the scalar variable  $h = \text{score} * (\text{theta} - \text{thetaref})$  for each point *theta* along the parameter grid. *model\_file* has to point to the filename of a saved *ScoreEstimator* instance. *hist\_bins* defines the binning of the histogram. *include\_xsec* sets whether the Poisson likelihood of the total number of events is included or not.

MadMiner calculates one p-value for every parameter point on an evenly spaced grid specified by *grid\_ranges* and *grid\_resolutions*. For instance, in a three-dimensional parameter space, *grid\_ranges*=[(-1., 1.), (-2., 2.), (-3., 3.)] and *grid\_resolutions*=[10,10,10] will start the calculation along  $10^3$  parameter points in a cube with edges (-1, 1) in the first parameter and so on.

### Parameters

#### **mode**

[{"rate", "histo", "ml", "sally", "sallino", "adaptive-sally"}] Defines how the likelihood ratio test statistic is calculated. See above.

#### **x\_observed**

[ndarray] Observed data with shape (*n\_events*, *n\_observables*). The observables have to be the same used throughout the MadMiner analysis, for instance specified in the *Delphes-Reader* class with *add\_observables*.



#### **grid\_ranges**

[list of (tuple of float) or None, optional] Specifies the boundaries of the parameter grid on which the p-values are evaluated. It should be  $[(min, max), (min, max), \dots, (min, max)]$ , where the list goes over all parameters and *min* and *max* are float. If None, *thetas\_eval* has to be given. Default: None.

#### **grid\_resolutions**

[int or list of int, optional] Resolution of the parameter space grid on which the p-values are evaluated. If int, the resolution is the same along every dimension of the hypercube. If list of int, the individual entries specify the number of points along each parameter individually. Doesn't have any effect if *grid\_ranges* is None. Default value: 25.

#### **include\_xsec**

[bool, optional] Whether the Poisson likelihood representing the total number of events is included in the analysis. Default value: True.

#### **model\_file**

[str or None, optional] Filename of a saved neural network estimating the likelihood, likelihood ratio, or score. Required if mode is anything except "rate" or "histo". Default value: None.

#### **hist\_vars**

[list of str or None, optional] Kinematic variables used in the histograms when mode is "histo". The names are the same as used for instance in *DelphesReader*. Default value: None.

#### **score\_components**

[None or list of int, optional] Defines the score components used when mode is "sally" or "adaptive-sally". Default value: None.

#### **hist\_bins**

[int or list of (int or ndarray) or None, optional] Defines the histogram binning when mode is "histo", "sally", "adaptive-sally", or "sallino". If int, gives the number of bins automatically chosen for each summary statistic. If list, each entry corresponds to one summary statistic (e.g. kinematic variable specified by *hist\_vars* or estimated score component); an int entry corresponds to the number of automatically chosen bins, an ndarray specifies the bin edges along this dimension explicitly. If None, the bins are chosen according to the defaults: for one summary statistic the default is 25 bins, for 2 it's 8 bins along each direction, for more it's 5 per dimension. Default value: None.

#### **thetaref**

[ndarray or None, optional] Defines the reference parameter point at which the score is evaluated for mode "sallino" or "adaptive-sally". If None, the origin in parameter space,  $[0., 0., \dots, 0.]$ , is used. Default value: None.

#### **luminosity**

[float, optional] Integrated luminosity in  $\text{pb}^{-1}$  assumed in the analysis. Default value: 300000.

#### **weighted\_histo**

[bool, optional] If True, the histograms used for the modes "histo", "sally", "sallino", and "adaptive-sally" use one set of weighted events to construct the histograms at every point along the parameter grid, only with different weights for each parameter point on the grid. If False, independent unweighted event samples are drawn for each parameter point on the grid. Default value: True.

#### **n\_histo\_toys**

[int or None, optional] Number of events drawn to construct the histograms used for the modes "histo", "sally", "sallino", and "adaptive-sally". If None and *weighted\_histo* is True,

all events in the training fraction of the MadMiner file are used. If `None` and `weighted_histo` is `False`, 100000 events are used. Default value: 100000.

**histo\_theta\_batchsize**

[int or `None`, optional] Number of histograms constructed in parallel for the modes “histo”, “sally”, “sallino”, and “adaptive-sally” and if `weighted_histo` is `True`. A larger number speeds up the calculation, but requires more memory. Default value: 1000.

**n\_observed**

[int or `None`, optional] If not `None`, the likelihood ratio is rescaled to this number of observed events before calculating p-values. Default value: `None`.

**dof**

[int or `None`, optional] If not `None`, sets the number of parameters for the calculation of the p-values. If `None`, the overall number of parameters is used. Default value: `None`.

**test\_split**

[float, optional] Fraction of weighted events in the MadMiner file reserved for evaluation. Default value: 0.2.

**return\_histos**

[bool, optional] If `True` and if mode is “histo”, “sally”, “adaptive-sally”, or “sallino”, the function returns histogram objects for each point along the grid.

**fix\_adaptive\_binning**

[[`False`, “center”, “grid”, “auto-grid”, “auto-center”], optional] If not `False` and if mode is “histo”, “sally”, “adaptive-sally”, or “sallino”, the automatic histogram binning is the same for every point along the parameter grid. For “center”, the central point in the parameter grid is used to determine the binning, for “grid” all points in the parameter grid are combined for this. For “auto-grid” or “auto-center”, this option is turned on if mode is “histo” or “sally”, but not for “adaptive-sally” or “sallino”. Default value: “auto-grid”.

**return\_observed**

[bool, optional] Whether the observed values of the summary statistics are returned. Default value: `False`.

**postprocessing**

[`None` or function] If not `None`, points to a function that processes the summary statistics before being fed into histograms. Default value: `None`.

**n\_binning\_toys**

[int or `None`] Number of toy events used to determine the binning of adaptive histograms. Default value: 100000.

**thetas\_eval**

[ndarray or `None`] Manually specifies the parameter point at which the likelihood and p-values are evaluated. If `None`, `grid_ranges` and `resolution` are used instead to construct a regular grid. Default value: `None`.

**Returns**

**parameter\_grid**

[ndarray] Parameter points at which the p-values are evaluated with shape (*n\_grid\_points*, *n\_parameters*).

**p\_values**

[ndarray] Observed p-values for each parameter point on the grid, with shape (*n\_grid\_points*,).

**mle**

[int] Index of the parameter point with the best fit (largest p-value / smallest -2 log likelihood

ratio).

**log\_likelihood\_ratio\_kin**

[ndarray or None] log likelihood ratio based only on kinematics for each point of the grid, with shape  $(n\_grid\_points,)$ .

**log\_likelihood\_rate**

[ndarray or None] log likelihood based only on the total rate for each point of the grid, with shape  $(n\_grid\_points,)$ .

**histos**

[None or list of Histogram] None if return\_histos is False. Otherwise a list of histogram objects for each point on the grid. This can be useful for debugging or for plotting the histograms.

## 12.3 Module contents



## MADMINER.ML PACKAGE

### 13.1 Submodules

### 13.2 `madminer.ml.base` module

**class** `madminer.ml.base.ConditionalEstimator`(*features=None, n\_hidden=(100,)*, *activation='tanh'*,  
*dropout\_prob=0.0*)

Bases: *Estimator*, ABC

Abstract class for estimator that is conditional on theta. Subclassed by *ParameterizedRatioEstimator*, *DoubleParameterizedRatioEstimator*, and *LikelihoodEstimator* (but not *ScoreEstimator*).

Adds functionality to rescale parameters.

#### Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(*args, **kwargs)</code>	Log likelihood ratio estimation.
<code>evaluate_score(*args, **kwargs)</code>	Score estimation.
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).

<code>evaluate</code>	
<code>initialize_input_transform</code>	
<code>initialize_parameter_transform</code>	
<code>train</code>	

**initialize\_parameter\_transform**(*theta, transform=True, overwrite=True*)

**load(filename)**

Loads a trained model from files.

**Parameters**

**filename**

[str] Path to the files. ‘\_settings.json’ and ‘\_state\_dict.pl’ will be added.

**Returns**

**None**

**save(filename, save\_model=False)**

Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).

**Parameters**

**filename**

[str] Path to the files. ‘\_settings.json’ and ‘\_state\_dict.pl’ will be added.

**save\_model**

[bool, optional] If True, the whole model is saved in addition to the state dict. This is not necessary for loading it again with Estimator.load(), but can be useful for debugging, for instance to plot the computational graph.

**Returns**

**None**

**class madminer.ml.base.Estimator(features=None, n\_hidden=(100,), activation='tanh', dropout\_prob=0.0)**

Bases: ABC

Abstract class for any ML estimator. Subclassed by ParameterizedRatioEstimator, DoubleParameterizedRatioEstimator, ScoreEstimator, and LikelihoodEstimator.

Each instance of this class represents one neural estimator. The most important functions are:

- *Estimator.train()* to train an estimator. The keyword *method* determines the inference technique and whether a class instance represents a single-parameterized likelihood ratio estimator, a doubly-parameterized likelihood ratio estimator, or a local score estimator.
- *Estimator.evaluate()* to evaluate the estimator.
- *Estimator.save()* to save the trained model to files.
- *Estimator.load()* to load the trained model from files.

Please see the tutorial for a detailed walk-through.

## Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(*args, **kwargs)</code>	Log likelihood ratio estimation.
<code>evaluate_score(*args, **kwargs)</code>	Score estimation.
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>train</b>	

**calculate\_fisher\_information**(*x*, *theta*=None, *weights*=None, *n\_events*=1, *sum\_events*=True)

Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.

### Parameters

**x**

[str or ndarray] Sample of observations, or path to numpy file with observations. Note that this sample has to be sampled from the reference parameter where the score is estimated with the SALLY / SALLINO estimator.

**theta: None or ndarray**

Numerator parameter point, or filename of a pickled numpy array. Has no effect for ScoreEstimator.

**weights**

[None or ndarray, optional] Weights for the observations. If None, all events are taken to have equal weight. Default value: None.

**n\_events**

[float, optional] Expected number of events for which the kinematic Fisher information should be calculated. Default value: 1.

**sum\_events**

[bool, optional] If True, the expected Fisher information summed over the events *x* is calculated. If False, the per-event Fisher information for each event is returned. Default value: True.

### Returns

**fisher\_information**

[ndarray] Expected kinematic Fisher information matrix with shape (*n\_events*, *n\_parameters*, *n\_parameters*) if *sum\_events* is False or (*n\_parameters*, *n\_parameters*) if *sum\_events* is True.

**abstract evaluate**(\*args, \*\*kwargs)

**abstract evaluate\_log\_likelihood(\*args, \*\*kwargs)**

Log likelihood estimation. Signature depends on the type of estimator. The first returned value is the log likelihood with shape  $(n\_thetas, n\_x)$ .

**abstract evaluate\_log\_likelihood\_ratio(\*args, \*\*kwargs)**

Log likelihood ratio estimation. Signature depends on the type of estimator. The first returned value is the log likelihood ratio with shape  $(n\_thetas, n\_x)$  or  $(n\_x)$ .

**abstract evaluate\_score(\*args, \*\*kwargs)**

Score estimation. Signature depends on the type of estimator. The only returned value is the score with shape  $(n\_x)$ .

**initialize\_input\_transform(x, transform=True, overwrite=True)**

**load(filename)**

Loads a trained model from files.

**Parameters**

**filename**

[str] Path to the files. ‘\_settings.json’ and ‘\_state\_dict.pl’ will be added.

**Returns**

None

**save(filename, save\_model=False)**

Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).

**Parameters**

**filename**

[str] Path to the files. ‘\_settings.json’ and ‘\_state\_dict.pl’ will be added.

**save\_model**

[bool, optional] If True, the whole model is saved in addition to the state dict. This is not necessary for loading it again with `Estimator.load()`, but can be useful for debugging, for instance to plot the computational graph.

**Returns**

None

**abstract train(\*args, \*\*kwargs)**

**exception madminer.ml.base.TheresAGoodReasonThisDoesntWork**

Bases: Exception

## 13.3 madminer.ml.double\_parameterized\_ratio module

```
class madminer.ml.double_parameterized_ratio.DoubleParameterizedRatioEstimator(features=None,
                                                                                  n_hidden=(100),
                                                                                  activa-
                                                                                  tion='tanh',
                                                                                  dropout_prob=0.0)
```



Bases: *ConditionalEstimator*

A neural estimator of the likelihood ratio as a function of the observation  $x$ , the numerator hypothesis  $\theta_0$ , and the denominator hypothesis  $\theta_1$ .

## Methods

<code>calculate_fisher_information(*args, **kwargs)</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(x, theta0, theta1)</code>	Evaluates the log likelihood ratio as a function of the observation $x$ , the numerator hypothesis $\theta_0$ , and the denominator hypothesis $\theta_1$ .
<code>evaluate_score(*args, **kwargs)</code>	Score estimation.
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code>train(method, x, y, theta0, theta1[, r_xz, ...])</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>initialize_parameter_transform</b>	

**calculate\_fisher\_information(\*args, \*\*kwargs)**

Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.

### Parameters

#### **x**

[str or ndarray] Sample of observations, or path to numpy file with observations. Note that this sample has to be sampled from the reference parameter where the score is estimated with the SALLY / SALLINO estimator.

#### **theta: None or ndarray**

Numerator parameter point, or filename of a pickled numpy array. Has no effect for ScoreEstimator.

#### **weights**

[None or ndarray, optional] Weights for the observations. If None, all events are taken to have equal weight. Default value: None.

#### **n\_events**

[float, optional] Expected number of events for which the kinematic Fisher information should be calculated. Default value: 1.

#### **sum\_events**

[bool, optional] If True, the expected Fisher information summed over the events  $x$  is calculated. If False, the per-event Fisher information for each event is returned. Default value: True.

## Returns

### **fisher\_information**

[ndarray] Expected kinematic Fisher information matrix with shape  $(n\_events, n\_parameters, n\_parameters)$  if `sum_events` is False or  $(n\_parameters, n\_parameters)$  if `sum_events` is True.

**evaluate**(\*args, \*\*kwargs)

**evaluate\_log\_likelihood**(\*args, \*\*kwargs)

Log likelihood estimation. Signature depends on the type of estimator. The first returned value is the log likelihood with shape  $(n\_thetas, n\_x)$ .

**evaluate\_log\_likelihood\_ratio**(x, theta0, theta1, test\_all\_combinations=True, evaluate\_score=False)

Evaluates the log likelihood ratio as a function of the observation x, the numerator hypothesis theta0, and the denominator hypothesis theta1.

## Parameters

### **x**

[str or ndarray] Observations or filename of a pickled numpy array.

### **theta0**

[ndarray or str] Numerator parameter points or filename of a pickled numpy array.

### **theta1**

[ndarray or str] Denominator parameter points or filename of a pickled numpy array.

### **test\_all\_combinations**

[bool, optional] If False, the number of samples in the observable and theta files has to match, and the likelihood ratio is evaluated only for the combinations  $r(x_i | \theta_{0_i}, \theta_{1_i})$ . If True,  $r(x_i | \theta_{0_j}, \theta_{1_j})$  for all pairwise combinations  $i, j$  are evaluated. Default value: True.

### **evaluate\_score**

[bool, optional] Sets whether in addition to the likelihood ratio the score is evaluated. Default value: False.

## Returns

### **log\_likelihood\_ratio**

[ndarray] The estimated log likelihood ratio. If `test_all_combinations` is True, the result has shape  $(n\_thetas, n\_x)$ . Otherwise, it has shape  $(n\_samples,)$ .

### **score0**

[ndarray or None] None if `evaluate_score` is False. Otherwise the derived estimated score at *theta0*. If `test_all_combinations` is True, the result has shape  $(n\_thetas, n\_x, n\_parameters)$ . Otherwise, it has shape  $(n\_samples, n\_parameters)$ .

### **score1**

[ndarray or None] None if `evaluate_score` is False. Otherwise the derived estimated score at *theta1*. If `test_all_combinations` is True, the result has shape  $(n\_thetas, n\_x, n\_parameters)$ . Otherwise, it has shape  $(n\_samples, n\_parameters)$ .

**evaluate\_score**(\*args, \*\*kwargs)

Score estimation. Signature depends on the type of estimator. The only returned value is the score with shape  $(n\_x)$ .

```
train(method, x, y, theta0, theta1, r_xz=None, t_xz0=None, t_xz1=None, x_val=None, y_val=None,
      theta0_val=None, theta1_val=None, r_xz_val=None, t_xz0_val=None, t_xz1_val=None, alpha=1.0,
      optimizer='amsgrad', n_epochs=50, batch_size=128, initial_lr=0.001, final_lr=0.0001,
      nesterov_momentum=None, validation_split=0.25, early_stopping=True, scale_inputs=True,
      shuffle_labels=False, limit_sample_size=None, memmap=False, verbose='some',
      scale_parameters=True, n_workers=8, clip_gradient=None, early_stopping_patience=None)
```

Trains the network.

### Parameters

#### **method**

[str] The inference method used for training. Allowed values are ‘alice’, ‘alices’, ‘carl’, ‘cascal’, ‘rascal’, and ‘rolr’.

#### **x**

[ndarray or str] Observations, or filename of a pickled numpy array.

#### **y**

[ndarray or str] Class labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array.

#### **theta0**

[ndarray or str] Numerator parameter point, or filename of a pickled numpy array.

#### **theta1**

[ndarray or str] Denominator parameter point, or filename of a pickled numpy array.

#### **r\_xz**

[ndarray or str or None, optional] Joint likelihood ratio, or filename of a pickled numpy array. Default value: None.

#### **t\_xz0**

[ndarray or str or None, optional] Joint scores at theta0, or filename of a pickled numpy array. Default value: None.

#### **t\_xz1**

[ndarray or str or None, optional] Joint scores at theta1, or filename of a pickled numpy array. Default value: None.

#### **x\_val**

[ndarray or str or None, optional] Validation observations, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### **y\_val**

[ndarray or str or None, optional] Validation labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### **theta0\_val**

[ndarray or str or None, optional] Validation numerator parameter points, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### **theta1\_val**

[ndarray or str or None, optional] Validation denominator parameter points, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### **r\_xz\_val**

[ndarray or str or None, optional] Validation joint likelihood ratio, or filename of a pickled

numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**t\_xz0\_val**

[ndarray or str or None, optional] Validation joint scores at `theta0`, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**t\_xz1\_val**

[ndarray or str or None, optional] Validation joint scores at `theta1`, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**alpha**

[float, optional] Hyperparameter weighting the score error in the loss function of the ‘alices’, ‘rascal’, and ‘cascal’ methods. Default value: 1.

**optimizer**

[{“adam”, “amsgrad”, “sgd”}, optional] Optimization algorithm. Default value: “amsgrad”.

**n\_epochs**

[int, optional] Number of epochs. Default value: 50.

**batch\_size**

[int, optional] Batch size. Default value: 128.

**initial\_lr**

[float, optional] Learning rate during the first epoch, after which it exponentially decays to `final_lr`. Default value: 0.001.

**final\_lr**

[float, optional] Learning rate during the last epoch. Default value: 0.0001.

**nesterov\_momentum**

[float or None, optional] If trainer is “sgd”, sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if `early_stopping` is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if `validation_split` is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels (`y`, `r_xz`, `t_xz`) are shuffled, while the observations (`x`) remain in their normal order. This serves as a closure test, in particular as cross-check against overfitting: an estimator trained with `shuffle_labels=True` should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{"all", "many", "some", "few", "none"}, optional] Determines verbosity of training. Default value: "some".

**Returns****result: ndarray**

Training and validation losses from `DoubleParameterizedRatioTrainer.train`

## 13.4 madminer.ml.ensemble module

**class** `madminer.ml.ensemble.Ensemble`(*estimators=None*)

Bases: `object`

Ensemble methods for likelihood, likelihood ratio, and score estimation.

Generally, Ensemble instances can be used very similarly to Estimator instances:

- The initialization of Ensemble takes a list of (trained or untrained) Estimator instances.
- The methods `Ensemble.train_one()` and `Ensemble.train_all()` train the estimators (this can also be done outside of Ensemble).
- `Ensemble.calculate_expectation()` can be used to calculate the expectation of the estimation likelihood ratio or the expected estimated score over a validation sample. Ideally (and assuming the correct sampling), these expectation values should be close to zero. Deviations from zero therefore point out that the estimator is probably inaccurate.
- `Ensemble.evaluate_log_likelihood()`, `Ensemble.evaluate_log_likelihood_ratio()`, `Ensemble.evaluate_score()`, and `Ensemble.calculate_fisher_information()` can then be used to calculate ensemble predictions.
- `Ensemble.save()` and `Ensemble.load()` can store all estimators in one folder.

The individual estimators in the ensemble can be trained with different methods, but they have to be of the same type: either all estimators are `ParameterizedRatioEstimator` instances, or all estimators are `DoubleParameterizedRatioEstimator` instances, or all estimators are `ScoreEstimator` instances, or all estimators are `LikelihoodEstimator` instances..

**Parameters****estimators**

[None or list of Estimator, optional] If int, sets the number of estimators that will be created as new `MLForge` instances. If list, sets the estimators directly, either from `MLForge` instances or filenames (that are then loaded with `MLForge.load()`). If None, the ensemble is initialized without estimators. Note that the estimators have to be consistent: either all of them are trained with a local score method ('sally' or 'sallino'); or all of them are trained with a single-parameterized method ('carl', 'rolr', 'rascal', 'scandal', 'alice', or 'alices'); or all of them are trained with a doubly parameterized method ('carl2', 'rolr2', 'rascal2', 'alice2', or 'alices2'). Mixing estimators of different types within one of these three categories is supported, but mixing estimators from different categories is not and will raise a `RuntimeException`. Default value: None.

**Attributes**

### estimators

[list of Estimator] The estimators in the form of MLForge instances.

## Methods

<code>add_estimator(estimator)</code>	Adds an estimator to the ensemble.
<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates expected Fisher information matrices for an ensemble of ScoreEstimator instances.
<code>evaluate_log_likelihood([estimator_weights, ...])</code>	Estimates the log likelihood from each estimator and returns the ensemble mean (and, if <code>calculate_covariance</code> is <code>True</code> , the covariance between them).
<code>evaluate_log_likelihood_ratio([...])</code>	Estimates the log likelihood ratio from each estimator and returns the ensemble mean (and, if <code>calculate_covariance</code> is <code>True</code> , the covariance between them).
<code>evaluate_score([estimator_weights, ...])</code>	Estimates the score from each estimator and returns the ensemble mean (and, if <code>calculate_covariance</code> is <code>True</code> , the covariance between them).
<code>load(folder)</code>	Loads the estimator ensemble from a folder.
<code>save(folder[, save_model])</code>	Saves the estimator ensemble to a folder.
<code>train_all(**kwargs)</code>	Trains all estimators.
<code>train_one(i, **kwargs)</code>	Trains an individual estimator.

### `add_estimator(estimator)`

Adds an estimator to the ensemble.

#### Parameters

##### **estimator**

[Estimator] The estimator.

#### Returns

None

**`calculate_fisher_information`**(*x*, *theta*=None, *obs\_weights*=None, *estimator\_weights*=None, *n\_events*=1, *mode*='score', *calculate\_covariance*=True, *sum\_events*=True, *epsilon\_shift*=0.001)

Calculates expected Fisher information matrices for an ensemble of ScoreEstimator instances.

There are two ways of calculating the ensemble average. In the default “score” mode, the ensemble average for the score is calculated for each event, and the Fisher information is calculated based on these mean scores. In the “information” mode, the Fisher information is calculated for each estimator separately and the ensemble mean is calculated only for the final Fisher information matrix. The “score” mode is generally assumed to be more precise and is the default.

In the “score” mode, the covariance matrix of the final result is calculated in the following way:

- For each event  $x$  and each estimator  $a$ , the “shifted” predicted score is calculated as  $t_a'(x) = t(x) + 1/\sqrt{n} * (t_a(x) - t(x))$ . Here  $t(x)$  is the mean score (averaged over the ensemble) for this event,  $t_a(x)$  is the prediction of estimator  $a$  for this event, and  $n$  is the number of estimators. The ensemble variance of these shifted score predictions is equal to the uncertainty on the mean of the ensemble of original predictions.

- For each estimator  $a$ , the shifted Fisher information matrix  $I_a'$  is calculated from the shifted predicted scores.
- The ensemble covariance between all Fisher information matrices  $I_a'$  is calculated and taken as the measure of uncertainty on the Fisher information calculated from the mean scores.

In the “information” mode, the user has the option to treat all estimators equally (‘committee method’) or to give those with expected score close to zero (as calculated by `calculate_expectation()`) a higher weight. In this case, the ensemble mean  $I$  is calculated as  $I = \sum_i w_i I_i$  with weights  $w_i = \exp(-\text{vote\_expectation\_weight} |E[t_i]|) / \sum_j \exp(-\text{vote\_expectation\_weight} |E[t_k]|)$ . Here  $I_i$  are the individual estimators and  $E[t_i]$  is the expectation value calculated by `calculate_expectation()`.

### Parameters

#### **x**

[str or ndarray] Sample of observations, or path to numpy file with observations, as saved by the `madminer.sampling.SampleAugmenter` functions. Note that this sample has to be sampled from the reference parameter where the score is estimated with the SALLY / SALLINO estimator!

#### **obs\_weights**

[None or ndarray, optional] Weights for the observations. If None, all events are taken to have equal weight. Default value: None.

#### **estimator\_weights**

[ndarray or None, optional] Weights for each estimator in the ensemble. If None, all estimators have an equal vote. Default value: None.

#### **n\_events**

[float, optional] Expected number of events for which the kinematic Fisher information should be calculated. Default value: 1.

#### **mode**

[{"score", "information"}, optional] If mode is “information”, the Fisher information for each estimator is calculated individually and only then are the sample mean and covariance calculated. If mode is “score”, the sample mean is calculated for the score for each event. Default value: “score”.

#### **calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: True.

#### **sum\_events**

[bool, optional] If True or mode is “information”, the expected Fisher information summed over the events  $x$  is calculated. If False and mode is “score”, the per-event Fisher information for each event is returned. Default value: True.

#### **epsilon\_shift**

[float, optional] Small numerical factor in the error propagation. Default value: 0.001.

### Returns

#### **mean\_prediction**

[ndarray] Expected kinematic Fisher information matrix with shape  $(n\_events, n\_parameters, n\_parameters)$  if `sum_events` is False and mode is “score”, or  $(n\_parameters, n\_parameters)$  in any other case.

#### **covariance**

[ndarray or None] The covariance of the estimated Fisher information matrix. This object has four indices,  $cov_{(ij)(i'j')}$ , ordered as  $i\ j\ i'\ j'$ . It has shape  $(n\_parameters, n\_parameters, n\_parameters, n\_parameters)$ .

**evaluate\_log\_likelihood**(*estimator\_weights=None, calculate\_covariance=False, \*\*kwargs*)

Estimates the log likelihood from each estimator and returns the ensemble mean (and, if calculate\_covariance is True, the covariance between them).

#### Parameters

##### **estimator\_weights**

[ndarray or None, optional] Weights for each estimator in the ensemble. If None, all estimators have an equal vote. Default value: None.

##### **calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: False.

##### **kwargs**

Arguments for the evaluation. See the documentation of the relevant Estimator class.

#### Returns

##### **log\_likelihood**

[ndarray] Mean prediction for the log likelihood.

##### **covariance**

[ndarray or None] If calculate\_covariance is True, the covariance matrix between the estimators. Otherwise None.

**evaluate\_log\_likelihood\_ratio**(*estimator\_weights=None, calculate\_covariance=False, \*\*kwargs*)

Estimates the log likelihood ratio from each estimator and returns the ensemble mean (and, if calculate\_covariance is True, the covariance between them).

#### Parameters

##### **estimator\_weights**

[ndarray or None, optional] Weights for each estimator in the ensemble. If None, all estimators have an equal vote. Default value: None.

##### **calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: False.

##### **kwargs**

Arguments for the evaluation. See the documentation of the relevant Estimator class.

#### Returns

##### **log\_likelihood\_ratio**

[ndarray] Mean prediction for the log likelihood ratio.

##### **covariance**

[ndarray or None] If calculate\_covariance is True, the covariance matrix between the estimators. Otherwise None.

**evaluate\_score**(*estimator\_weights=None, calculate\_covariance=False, \*\*kwargs*)

Estimates the score from each estimator and returns the ensemble mean (and, if calculate\_covariance is True, the covariance between them).

#### Parameters

##### **estimator\_weights**

[ndarray or None, optional] Weights for each estimator in the ensemble. If None, all estimators have an equal vote. Default value: None.



**calculate\_covariance**

[bool, optional] If True, the covariance between the different estimators is calculated. Default value: False.

**kwargs**

Arguments for the evaluation. See the documentation of the relevant Estimator class.

**Returns****log\_likelihood\_ratio**

[ndarray] Mean prediction for the log likelihood ratio.

**covariance**

[ndarray or None] If calculate\_covariance is True, the covariance matrix between the estimators. Otherwise None.

**load(folder)**

Loads the estimator ensemble from a folder.

**Parameters****folder**

[str] Path to the folder.

**Returns**

None

**save(folder, save\_model=False)**

Saves the estimator ensemble to a folder.

**Parameters****folder**

[str] Path to the folder.

**save\_model**

[bool, optional] If True, the whole model is saved in addition to the state dict. This is not necessary for loading it again with Ensemble.load(), but can be useful for debugging, for instance to plot the computational graph.

**Returns**

None

**train\_all(\*\*kwargs)**

Trains all estimators. See *Estimator.train()*.

**Parameters****kwargs**

[dict] Parameters for *Estimator.train()*. If a value in this dict is a list, it has to have length *n\_estimators* and contain one value of this parameter for each of the estimators. Otherwise the value is used as parameter for the training of all the estimators.

**Returns****result\_list: list of ndarray**

List of training and validation losses from estimator training

**train\_one(i, \*\*kwargs)**

Trains an individual estimator. See *Estimator.train()*.

**Parameters**

**i**[int] The index  $0 \leq i < n\_estimators$  of the estimator to be trained.**kwargs**[dict] Parameters for *Estimator.train()*.**Returns****result: ndarray**

Training and validation losses from estimator training

## 13.5 madminer.ml.likelihood module

```
class madminer.ml.likelihood.LikelihoodEstimator(features=None, n_components=1, n_mades=5,
                                                  n_hidden=(100,), activation='tanh',
                                                  batch_norm=None)
```

Bases: *ConditionalEstimator*

A neural estimator of the density or likelihood evaluated at a reference hypothesis as a function of the observation  $x$ .

**Parameters****features**

[list of int or None, optional] Indices of observables (features) that are used as input to the neural networks. If None, all observables are used. Default value: None.

**n\_components**

[int, optional] The number of Gaussian base components in a MADE MoG. If 1, a plain MADE is used. Default value: 1.

**n\_mades**

[int, optional] The number of MADE layers. Default value: 3.

**n\_hidden**

[tuple of int, optional] Units in each hidden layer in the neural networks. If method is 'nde' or 'scandal', this refers to the setup of each individual MADE layer. Default value: (100,).

**activation**

[{ 'tanh', 'sigmoid', 'relu' }, optional] Activation function. Default value: 'tanh'.

**batch\_norm**

[None or float, optional] If not None, batch normalization is used, where this value sets the alpha parameter in the calculation of the running average of the mean and variance. Default value: None.

## Methods

<code>calculate_fisher_information(*args, **kwargs)</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(x, theta[, ...])</code>	Evaluates the log likelihood as a function of the observation <i>x</i> and the parameter point <i>theta</i> .
<code>evaluate_log_likelihood_ratio(x, theta0, ...)</code>	Evaluates the log likelihood ratio as a function of the observation <i>x</i> , the numerator parameter point <i>theta0</i> , and the denominator parameter point <i>theta1</i> .
<code>evaluate_score(*args, **kwargs)</code>	Score estimation.
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code>train(method, x, theta[, t_xz, x_val, ...])</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>initialize_parameter_transform</b>	

### `calculate_fisher_information(*args, **kwargs)`

Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.

#### Parameters

##### **x**

[str or ndarray] Sample of observations, or path to numpy file with observations. Note that this sample has to be sampled from the reference parameter where the score is estimated with the SALLY / SALLINO estimator.

##### **theta: None or ndarray**

Numerator parameter point, or filename of a pickled numpy array. Has no effect for ScoreEstimator.

##### **weights**

[None or ndarray, optional] Weights for the observations. If None, all events are taken to have equal weight. Default value: None.

##### **n\_events**

[float, optional] Expected number of events for which the kinematic Fisher information should be calculated. Default value: 1.

##### **sum\_events**

[bool, optional] If True, the expected Fisher information summed over the events *x* is calculated. If False, the per-event Fisher information for each event is returned. Default value: True.

#### Returns

##### **fisher\_information**

[ndarray] Expected kinematic Fisher information matrix with shape  $(n\_events, n\_parameters, n\_parameters)$  if *sum\_events* is False or  $(n\_parameters, n\_parameters)$  if *sum\_events* is True.

**evaluate**(\*args, \*\*kwargs)

**evaluate\_log\_likelihood**(x, theta, test\_all\_combinations=True, evaluate\_score=False)

Evaluates the log likelihood as a function of the observation x and the parameter point theta.

#### Parameters

**x**

[ndarray or str] Sample of observations, or path to numpy file with observations.

**theta**

[ndarray or str] Parameter points, or path to numpy file with parameter points.

**test\_all\_combinations**

[bool, optional] If method is not ‘sally’ and not ‘sallino’: If False, the number of samples in the observable and theta files has to match, and the likelihood ratio is evaluated only for the combinations  $r(x_i | \theta_{0,i}, \theta_{1,i})$ . If True,  $r(x_i | \theta_{0,j}, \theta_{1,j})$  for all pairwise combinations  $i, j$  are evaluated. Default value: True.

**evaluate\_score**

[bool, optional] If method is not ‘sally’ and not ‘sallino’, this sets whether in addition to the likelihood ratio the score is evaluated. Default value: False.

#### Returns

**log\_likelihood**

[ndarray] The estimated log likelihood. If test\_all\_combinations is True, the result has shape  $(n_{\theta}, n_x)$ . Otherwise, it has shape  $(n_{\text{samples}},)$ .

**score**

[ndarray or None] None if evaluate\_score is False. Otherwise the derived estimated score at theta. If test\_all\_combinations is True, the result has shape  $(n_{\theta}, n_x, n_{\text{parameters}})$ . Otherwise, it has shape  $(n_{\text{samples}}, n_{\text{parameters}})$ .

**evaluate\_log\_likelihood\_ratio**(x, theta0, theta1, test\_all\_combinations, evaluate\_score=False)

Evaluates the log likelihood ratio as a function of the observation x, the numerator parameter point theta0, and the denominator parameter point theta1.

#### Parameters

**x**

[ndarray or str] Sample of observations, or path to numpy file with observations.

**theta0**

[ndarray or str] Numerator parameters, or path to numpy file.

**theta1**

[ndarray or str] Denominator parameters, or path to numpy file.

**test\_all\_combinations**

[bool, optional] If method is not ‘sally’ and not ‘sallino’: If False, the number of samples in the observable and theta files has to match, and the likelihood ratio is evaluated only for the combinations  $r(x_i | \theta_{0,i}, \theta_{1,i})$ . If True,  $r(x_i | \theta_{0,j}, \theta_{1,j})$  for all pairwise combinations  $i, j$  are evaluated. Default value: True.

**evaluate\_score**

[bool, optional] If method is not ‘sally’ and not ‘sallino’, this sets whether in addition to the likelihood ratio the score is evaluated. Default value: False.

#### Returns

**log\_likelihood**

[ndarray] The estimated log likelihood. If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x)$ . Otherwise, it has shape  $(n\_samples,)$ .

**score**

[ndarray or None] None if `evaluate_score` is `False`. Otherwise the derived estimated score at *theta*. If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x, n\_parameters)$ . Otherwise, it has shape  $(n\_samples, n\_parameters)$ .

**evaluate\_score**(*\*args, \*\*kwargs*)

Score estimation. Signature depends on the type of estimator. The only returned value is the score with shape  $(n\_x)$ .

**train**(*method, x, theta, t\_xz=None, x\_val=None, theta\_val=None, t\_xz\_val=None, alpha=1.0, optimizer='amsgrad', n\_epochs=50, batch\_size=128, initial\_lr=0.001, final\_lr=0.0001, nesterov\_momentum=None, validation\_split=0.25, early\_stopping=True, scale\_inputs=True, shuffle\_labels=False, limit\_samplesize=None, memmap=False, verbose='some', scale\_parameters=True, n\_workers=8, clip\_gradient=None, early\_stopping\_patience=None*)

Trains the network.

**Parameters****method**

[str] The inference method used for training. Allowed values are ‘nde’ and ‘scandal’.

**x**

[ndarray or str] Observations, or filename of a pickled numpy array.

**theta**

[ndarray or str] Numerator parameter point, or filename of a pickled numpy array.

**t\_xz**

[ndarray or str or None, optional] Joint scores at theta, or filename of a pickled numpy array. Default value: None.

**x\_val**

[ndarray or str or None, optional] Validation observations, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**theta\_val**

[ndarray or str or None, optional] Validation numerator parameter points, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**t\_xz\_val**

[ndarray or str or None, optional] Validation joint scores at theta, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**alpha**

[float, optional] Hyperparameter weighting the score error in the loss function of the ‘alices’, ‘rascal’, and ‘cascal’ methods. Default value: 1.

**optimizer**

[{“adam”, “amsgrad”, “sgd”}, optional] Optimization algorithm. Default value: “amsgrad”.

**n\_epochs**

[int, optional] Number of epochs. Default value: 50.

**batch\_size**

[int, optional] Batch size. Default value: 128.

**initial\_lr**

[float, optional] Learning rate during the first epoch, after which it exponentially decays to final\_lr. Default value: 0.001.

**final\_lr**

[float, optional] Learning rate during the last epoch. Default value: 0.0001.

**nesterov\_momentum**

[float or None, optional] If trainer is “sgd”, sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if early\_stopping is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if validation\_split is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels ( $y$ ,  $r_{xz}$ ,  $t_{xz}$ ) are shuffled, while the observations ( $x$ ) remain in their normal order. This serves as a closure test, in particular as cross-check against overfitting: an estimator trained with shuffle\_labels=True should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{"all", "many", "some", "few", "none"}, optional] Determines verbosity of training. Default value: “some”.

**scale\_parameters**

[bool, optional] Whether parameters are rescaled to mean zero and unit variance before going into the neural network. Default value: True.

**Returns**

**result: ndarray**

Training and validation losses from FlowTrainer.train

## 13.6 madminer.ml.lookup module

`madminer.ml.lookup.load_estimator(filename)`

## 13.7 madminer.ml.morphing\_aware module

**class** `madminer.ml.morphing_aware.MorphingAwareRatioEstimator`(*morphing\_setup\_filename=None*,  
*optimize\_morphing\_basis=False*,  
*features=None*, *n\_hidden=(100,)*,  
*activation='tanh'*,  
*dropout\_prob=0.0*)

Bases: *ParameterizedRatioEstimator*

### Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(x, theta[, ...])</code>	Evaluates the log likelihood ratio for given observations <i>x</i> between the given parameter point <i>theta</i> and the reference hypothesis.
<code>evaluate_log_likelihood_ratio_torch(x, theta)</code>	Evaluates the log likelihood ratio for given observations <i>x</i> between the given parameter point <i>theta</i> and the reference hypothesis.
<code>evaluate_score(x, theta[, nuisance_model])</code>	Evaluates the scores for given observations <i>x</i> between at a given parameter point <i>theta</i> .
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code>train(*args, **kwargs)</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>initialize_parameter_transform</b>	

**train**(\*args, \*\*kwargs)

Trains the network.

#### Parameters

##### method

[str] The inference method used for training. Allowed values are ‘alice’, ‘alices’, ‘carl’, ‘cascal’, ‘rascal’, and ‘rolr’.

##### x

[ndarray or str] Observations, or filename of a pickled numpy array.

- y**  
[ndarray or str] Class labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array.
- theta**  
[ndarray or str] Numerator parameter point, or filename of a pickled numpy array.
- r\_xz**  
[ndarray or str or None, optional] Joint likelihood ratio, or filename of a pickled numpy array. Default value: None.
- t\_xz**  
[ndarray or str or None, optional] Joint scores at theta, or filename of a pickled numpy array. Default value: None.
- x\_val**  
[ndarray or str or None, optional] Validation observations, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.
- y\_val**  
[ndarray or str or None, optional] Validation labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.
- theta\_val**  
[ndarray or str or None, optional] Validation numerator parameter points, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.
- r\_xz\_val**  
[ndarray or str or None, optional] Validation joint likelihood ratio, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.
- t\_xz\_val**  
[ndarray or str or None, optional] Validation joint scores at theta, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.
- alpha**  
[float, optional] Hyperparameter weighting the score error in the loss function of the ‘alices’, ‘rascal’, and ‘cascal’ methods. Default value: 1.
- optimizer**  
[{"adam", "amsgrad", "sgd"}, optional] Optimization algorithm. Default value: “amsgrad”.
- n\_epochs**  
[int, optional] Number of epochs. Default value: 50.
- batch\_size**  
[int, optional] Batch size. Default value: 128.
- initial\_lr**  
[float, optional] Learning rate during the first epoch, after which it exponentially decays to final\_lr. Default value: 0.001.
- final\_lr**  
[float, optional] Learning rate during the last epoch. Default value: 0.0001.



**nesterov\_momentum**

[float or None, optional] If trainer is “sgd”, sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if early\_stopping is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if validation\_split is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels ( $y$ ,  $r_{xz}$ ,  $t_{xz}$ ) are shuffled, while the observations ( $x$ ) remain in their normal order. This serves as a closure test, in particular as cross-check against overfitting: an estimator trained with shuffle\_labels=True should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{“all”, “many”, “some”, “few”, “none”, optional] Determines verbosity of training. Default value: “some”.

**scale\_parameters**

[bool, optional] Whether parameters are rescaled to mean zero and unit variance before going into the neural network. Default value: True.

**Returns**

**result: ndarray**

Training and validation losses from SingleParameterizedRatioTrainer.train or DoubleParameterizedRatioTrainer.train for example

```
class madminer.ml.morphing_aware.QuadraticMorphingAwareRatioEstimator(features=None,
                                                                    n_hidden=(100,),
                                                                    activation='tanh',
                                                                    dropout_prob=0.0)
```

Bases: *ParameterizedRatioEstimator*

Specific morphing-aware likelihood ratio estimator for a single parameter and  $\theta_1 = 0$ .

Uses the quadratic parameterization of 2007.10356:  $\hat{r}(x, \theta) = (1 + \theta A(x))^2 + (\theta B(x))^2$ .

## Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(x, theta[, ...])</code>	Evaluates the log likelihood ratio for given observations <i>x</i> between the given parameter point <i>theta</i> and the reference hypothesis.
<code>evaluate_log_likelihood_ratio_torch(x, theta)</code>	Evaluates the log likelihood ratio for given observations <i>x</i> between the given parameter point <i>theta</i> and the reference hypothesis.
<code>evaluate_score(x, theta[, nuisance_model])</code>	Evaluates the scores for given observations <i>x</i> between at a given parameter point <i>theta</i> .
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code><i>train</i>(*args, **kwargs)</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>initialize_parameter_transform</b>	

**`train(*args, **kwargs)`**

Trains the network.

### Parameters

#### **method**

[str] The inference method used for training. Allowed values are ‘alice’, ‘alices’, ‘carl’, ‘cascal’, ‘rascal’, and ‘rolr’.

#### **x**

[ndarray or str] Observations, or filename of a pickled numpy array.

#### **y**

[ndarray or str] Class labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array.

#### **theta**

[ndarray or str] Numerator parameter point, or filename of a pickled numpy array.

#### **r\_xz**

[ndarray or str or None, optional] Joint likelihood ratio, or filename of a pickled numpy array. Default value: None.

#### **t\_xz**

[ndarray or str or None, optional] Joint scores at *theta*, or filename of a pickled numpy array. Default value: None.

#### **x\_val**

[ndarray or str or None, optional] Validation observations, or filename of a pickled numpy array. If None and `validation_split > 0`, validation data will be randomly selected from the training data. Default value: None.

**y\_val**

[ndarray or str or None, optional] Validation labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

**theta\_val**

[ndarray or str or None, optional] Validation numerator parameter points, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

**r\_xz\_val**

[ndarray or str or None, optional] Validation joint likelihood ratio, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

**t\_xz\_val**

[ndarray or str or None, optional] Validation joint scores at theta, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

**alpha**

[float, optional] Hyperparameter weighting the score error in the loss function of the ‘alices’, ‘rascal’, and ‘cascal’ methods. Default value: 1.

**optimizer**

[{"adam", "amsgrad", "sgd"}, optional] Optimization algorithm. Default value: “amsgrad”.

**n\_epochs**

[int, optional] Number of epochs. Default value: 50.

**batch\_size**

[int, optional] Batch size. Default value: 128.

**initial\_lr**

[float, optional] Learning rate during the first epoch, after which it exponentially decays to final\_lr. Default value: 0.001.

**final\_lr**

[float, optional] Learning rate during the last epoch. Default value: 0.0001.

**nesterov\_momentum**

[float or None, optional] If trainer is “sgd”, sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if early\_stopping is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if validation\_split is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels (y, r\_xz, t\_xz) are shuffled, while the observations (x) remain in their normal order. This serves as a closure test, in particular as cross-check

against overfitting: an estimator trained with `shuffle_labels=True` should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{"all", "many", "some", "few", "none"}, optional] Determines verbosity of training. Default value: "some".

**scale\_parameters**

[bool, optional] Whether parameters are rescaled to mean zero and unit variance before going into the neural network. Default value: True.

**Returns****result: ndarray**

Training and validation losses from `SingleParameterizedRatioTrainer.train` or `DoubleParameterizedRatioTrainer.train` for example

## 13.8 `madminer.ml.parameterized_ratio` module

```
class madminer.ml.parameterized_ratio.ParameterizedRatioEstimator(features=None,  
                                                                n_hidden=(100,),  
                                                                activation='tanh',  
                                                                dropout_prob=0.0)
```

Bases: *ConditionalEstimator*

A neural estimator of the likelihood ratio as a function of the observation  $x$  as well as the numerator hypothesis  $\theta$ . The reference (denominator) hypothesis is kept fixed at some reference value and NOT modeled by the network.

## Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(x, theta[, ...])</code>	Evaluates the log likelihood ratio for given observations $x$ between the given parameter point $\theta$ and the reference hypothesis.
<code>evaluate_log_likelihood_ratio_torch(x, theta)</code>	Evaluates the log likelihood ratio for given observations $x$ between the given parameter point $\theta$ and the reference hypothesis.
<code>evaluate_score(x, theta[, nuisance_model])</code>	Evaluates the scores for given observations $x$ between at a given parameter point $\theta$ .
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code>train(method, x, y, theta[, r_xz, t_xz, ...])</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	
<b>initialize_parameter_transform</b>	

**evaluate**(\*args, \*\*kwargs)

**evaluate\_log\_likelihood**(\*args, \*\*kwargs)

Log likelihood estimation. Signature depends on the type of estimator. The first returned value is the log likelihood with shape  $(n\_thetas, n\_x)$ .

**evaluate\_log\_likelihood\_ratio**( $x, \theta, test\_all\_combinations=True, evaluate\_score=False$ )

Evaluates the log likelihood ratio for given observations  $x$  between the given parameter point  $\theta$  and the reference hypothesis.

### Parameters

**x**

[str or ndarray] Observations or filename of a pickled numpy array.

**theta**

[ndarray or str] Parameter points or filename of a pickled numpy array.

**test\_all\_combinations**

[bool, optional] If False, the number of samples in the observable and theta files has to match, and the likelihood ratio is evaluated only for the combinations  $r(x_i | \theta_{0_i}, \theta_{1_i})$ . If True,  $r(x_i | \theta_{0_j}, \theta_{1_j})$  for all pairwise combinations  $i, j$  are evaluated. Default value: True.

**evaluate\_score**

[bool, optional] Sets whether in addition to the likelihood ratio the score is evaluated. Default value: False.

### Returns

**log\_likelihood\_ratio**

[ndarray] The estimated log likelihood ratio. If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x)$ . Otherwise, it has shape  $(n\_samples,)$ .

**score**

[ndarray or None] None if `evaluate_score` is `False`. Otherwise the derived estimated score at  $\theta_0$ . If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x, n\_parameters)$ . Otherwise, it has shape  $(n\_samples, n\_parameters)$ .

**evaluate\_log\_likelihood\_ratio\_torch**(*x, theta, test\_all\_combinations=True*)

Evaluates the log likelihood ratio for given observations *x* between the given parameter point *theta* and the reference hypothesis.

**Parameters**

**x**

[torch.tensor] Observations.

**theta**

[torch.tensor] Parameter points.

**test\_all\_combinations**

[bool, optional] If `False`, the number of samples in the observable and *theta* files has to match, and the likelihood ratio is evaluated only for the combinations  $r(x_i | \theta_{0,i}, \theta_{1,i})$ . If `True`,  $r(x_i | \theta_{0,j}, \theta_{1,j})$  for all pairwise combinations *i, j* are evaluated. Default value: `True`.

**Returns**

**log\_likelihood\_ratio**

[torch.tensor] The estimated log likelihood ratio. If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x)$ . Otherwise, it has shape  $(n\_samples,)$ .

**evaluate\_score**(*x, theta, nuisance\_mode='keep'*)

Evaluates the scores for given observations *x* between at a given parameter point *theta*.

**Parameters**

**x**

[str or ndarray] Observations or filename of a pickled numpy array.

**theta**

[ndarray or str] Parameter points or filename of a pickled numpy array.

**nuisance\_mode**

[{"auto", "keep", "profile", "project"}] Decides how nuisance parameters are treated. If `nuisance_mode` is `"keep"`, the returned score is always  $(n+k)$ -dimensional.

**Returns**

**score**

[ndarray or None] The estimated score at *theta*. If `test_all_combinations` is `True`, the result has shape  $(n\_thetas, n\_x, n\_parameters)$ . Otherwise, it has shape  $(n\_samples, n\_parameters)$ .

**train**(*method, x, y, theta, r\_xz=None, t\_xz=None, x\_val=None, y\_val=None, theta\_val=None, r\_xz\_val=None, t\_xz\_val=None, alpha=1.0, optimizer='amsgrad', n\_epochs=50, batch\_size=128, initial\_lr=0.001, final\_lr=0.0001, nesterov\_momentum=None, validation\_split=0.25, early\_stopping=True, scale\_inputs=True, shuffle\_labels=False, limit\_samplesize=None, memmap=False, verbose='some', scale\_parameters=True, n\_workers=8, clip\_gradient=None, early\_stopping\_patience=None*)

Trains the network.

### Parameters

#### method

[str] The inference method used for training. Allowed values are ‘alice’, ‘alices’, ‘carl’, ‘cascal’, ‘rascal’, and ‘rolr’.

#### x

[ndarray or str] Observations, or filename of a pickled numpy array.

#### y

[ndarray or str] Class labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array.

#### theta

[ndarray or str] Numerator parameter point, or filename of a pickled numpy array.

#### r\_xz

[ndarray or str or None, optional] Joint likelihood ratio, or filename of a pickled numpy array. Default value: None.

#### t\_xz

[ndarray or str or None, optional] Joint scores at theta, or filename of a pickled numpy array. Default value: None.

#### x\_val

[ndarray or str or None, optional] Validation observations, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### y\_val

[ndarray or str or None, optional] Validation labels (0 = numerator, 1 = denominator), or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### theta\_val

[ndarray or str or None, optional] Validation numerator parameter points, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### r\_xz\_val

[ndarray or str or None, optional] Validation joint likelihood ratio, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### t\_xz\_val

[ndarray or str or None, optional] Validation joint scores at theta, or filename of a pickled numpy array. If None and validation\_split > 0, validation data will be randomly selected from the training data. Default value: None.

#### alpha

[float, optional] Hyperparameter weighting the score error in the loss function of the ‘alices’, ‘rascal’, and ‘cascal’ methods. Default value: 1.

#### optimizer

[{“adam”, “amsgrad”, “sgd”}, optional] Optimization algorithm. Default value: “amsgrad”.

**n\_epochs**

[int, optional] Number of epochs. Default value: 50.

**batch\_size**

[int, optional] Batch size. Default value: 128.

**initial\_lr**

[float, optional] Learning rate during the first epoch, after which it exponentially decays to final\_lr. Default value: 0.001.

**final\_lr**

[float, optional] Learning rate during the last epoch. Default value: 0.0001.

**nesterov\_momentum**

[float or None, optional] If trainer is “sgd”, sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if early\_stopping is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if validation\_split is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels ( $y$ ,  $r_{xz}$ ,  $t_{xz}$ ) are shuffled, while the observations ( $x$ ) remain in their normal order. This serves as a closure test, in particular as cross-check against overfitting: an estimator trained with shuffle\_labels=True should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{“all”, “many”, “some”, “few”, “none”, optional] Determines verbosity of training. Default value: “some”.

**scale\_parameters**

[bool, optional] Whether parameters are rescaled to mean zero and unit variance before going into the neural network. Default value: True.

**Returns**

**result: ndarray**

Training and validation losses from SingleParameterizedRatioTrainer.train or DoubleParameterizedRatioTrainer.train for example



## 13.9 madminer.ml.score module

**class** madminer.ml.score.ScoreEstimator(*features=None, n\_hidden=(100,), activation='tanh', dropout\_prob=0.0*)

Bases: *Estimator*

A neural estimator of the score evaluated at a fixed reference hypothesis as a function of the observation *x*.

### Parameters

#### features

[list of int or None, optional] Indices of observables (features) that are used as input to the neural networks. If None, all observables are used. Default value: None.

#### n\_hidden

[tuple of int, optional] Units in each hidden layer in the neural networks. If method is 'nde' or 'scandal', this refers to the setup of each individual MADE layer. Default value: (100,).

#### activation

[{'tanh', 'sigmoid', 'relu'}, optional] Activation function. Default value: 'tanh'.

### Methods

<code>calculate_fisher_information(x[, theta, ...])</code>	Calculates the expected Fisher information matrix based on the kinematic information in a given number of events.
<code>evaluate_log_likelihood(*args, **kwargs)</code>	Log likelihood estimation.
<code>evaluate_log_likelihood_ratio(*args, **kwargs)</code>	Log likelihood ratio estimation.
<code>evaluate_score(x[, theta, nuisance_model])</code>	Evaluates the score.
<code>load(filename)</code>	Loads a trained model from files.
<code>save(filename[, save_model])</code>	Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).
<code>set_nuisance(fisher_information, ...)</code>	Prepares the calculation of profiled scores, see <a href="https://arxiv.org/pdf/1903.01473.pdf">https://arxiv.org/pdf/1903.01473.pdf</a> .
<code>train(method, x, t_xz[, x_val, t_xz_val, ...])</code>	Trains the network.

<b>evaluate</b>	
<b>initialize_input_transform</b>	

**evaluate**(\*args, \*\*kwargs)

**evaluate\_log\_likelihood**(\*args, \*\*kwargs)

Log likelihood estimation. Signature depends on the type of estimator. The first returned value is the log likelihood with shape (*n\_thetas*, *n\_x*).

**evaluate\_log\_likelihood\_ratio**(\*args, \*\*kwargs)

Log likelihood ratio estimation. Signature depends on the type of estimator. The first returned value is the log likelihood ratio with shape (*n\_thetas*, *n\_x*) or (*n\_x*).

**evaluate\_score**(*x*, *theta=None*, *nuisance\_mode='auto'*)

Evaluates the score.

**Parameters**

**x**

[str or ndarray] Observations, or filename of a pickled numpy array.

**theta: None or ndarray, optional**

Has no effect for ScoreEstimator. Introduced just for conformity with other Estimators.

**nuisance\_mode**

[{"auto", "keep", "profile", "project"}] Decides how nuisance parameters are treated. If nuisance\_mode is "auto", the returned score is the (n+k)- dimensional score in the space of n parameters of interest and k nuisance parameters if *set\_profiling* has not been called, and the n-dimensional profiled score in the space of the parameters of interest if it has been called. For "keep", the returned score is always (n+k)-dimensional. For "profile", it is the n-dimensional profiled score. For "project", it is the n-dimensional projected score, i.e. ignoring the nuisance parameters.

**Returns**

**score**

[ndarray] Estimated score with shape (*n\_observations*, *n\_parameters*).

**load**(*filename*)

Loads a trained model from files.

**Parameters**

**filename**

[str] Path to the files. '\_settings.json' and '\_state\_dict.pl' will be added.

**Returns**

**None**

**save**(*filename*, *save\_model=False*)

Saves the trained model to four files: a JSON file with the settings, a pickled pyTorch state dict file, and numpy files for the mean and variance of the inputs (used for input scaling).

**Parameters**

**filename**

[str] Path to the files. '\_settings.json' and '\_state\_dict.pl' will be added.

**save\_model**

[bool, optional] If True, the whole model is saved in addition to the state dict. This is not necessary for loading it again with Estimator.load(), but can be useful for debugging, for instance to plot the computational graph.

**Returns**

**None**

**set\_nuisance**(*fisher\_information*, *parameters\_of\_interest*)

Prepares the calculation of profiled scores, see <https://arxiv.org/pdf/1903.01473.pdf>.

**Parameters**

**fisher\_information**

[ndarray] Fisher information with shape (*n\_parameters*, *n\_parameters*).

**parameters\_of\_interest**

[list of int] List of int, with  $0 \leq \text{remaining\_components}[i] < n\_parameters$ . Denotes which parameters are kept in the profiling, and their new order.

**Returns**

None

**train**(*method*, *x*, *t\_xz*, *x\_val*=None, *t\_xz\_val*=None, *optimizer*='amsgrad', *n\_epochs*=50, *batch\_size*=128, *initial\_lr*=0.001, *final\_lr*=0.0001, *nesterov\_momentum*=None, *validation\_split*=0.25, *early\_stopping*=True, *scale\_inputs*=True, *shuffle\_labels*=False, *limit\_sample\_size*=None, *memmap*=False, *verbose*='some', *n\_workers*=8, *clip\_gradient*=None, *early\_stopping\_patience*=None)

Trains the network.

**Parameters****method**

[str] The inference method used for training. Currently values are 'sally' and 'sallino', but at the training stage they are identical. So right now it doesn't matter which one you use.

**x**

[ndarray or str] Path to an unweighted sample of observations, as saved by the *madminer.sampling.SampleAugmenter* functions. Required for all inference methods.

**t\_xz**

[ndarray or str] Joint scores at the reference hypothesis, or filename of a pickled numpy array.

**optimizer**

[{"adam", "amsgrad", "sgd"}, optional] Optimization algorithm. Default value: "amsgrad".

**n\_epochs**

[int, optional] Number of epochs. Default value: 50.

**batch\_size**

[int, optional] Batch size. Default value: 128.

**initial\_lr**

[float, optional] Learning rate during the first epoch, after which it exponentially decays to *final\_lr*. Default value: 0.001.

**final\_lr**

[float, optional] Learning rate during the last epoch. Default value: 0.0001.

**nesterov\_momentum**

[float or None, optional] If trainer is "sgd", sets the Nesterov momentum. Default value: None.

**validation\_split**

[float or None, optional] Fraction of samples used for validation and early stopping (if *early\_stopping* is True). If None, the entire sample is used for training and early stopping is deactivated. Default value: 0.25.

**early\_stopping**

[bool, optional] Activates early stopping based on the validation loss (only if *validation\_split* is not None). Default value: True.

**scale\_inputs**

[bool, optional] Scale the observables to zero mean and unit variance. Default value: True.

**shuffle\_labels**

[bool, optional] If True, the labels ( $y$ ,  $r_{xz}$ ,  $t_{xz}$ ) are shuffled, while the observations ( $x$ ) remain in their normal order. This serves as a closure test, in particular as cross-check against overfitting: an estimator trained with `shuffle_labels=True` should predict to likelihood ratios around 1 and scores around 0.

**limit\_samplesize**

[int or None, optional] If not None, only this number of samples (events) is used to train the estimator. Default value: None.

**memmap**

[bool, optional.] If True, training files larger than 1 GB will not be loaded into memory at once. Default value: False.

**verbose**

[{"all", "many", "some", "few", "none"}, optional] Determines verbosity of training. Default value: "some".

**Returns**

**result: ndarray**

Training and validation losses from `LocalScoreTrainer.train`

## 13.10 Module contents

## MADMINER.PLOTTING PACKAGE

### 14.1 Submodules

### 14.2 `madminer.plotting.distributions` module

```
madminer.plotting.distributions.plot_distributions(filename, observables=None,
                                                  parameter_points=None,
                                                  uncertainties='nuisance',
                                                  nuisance_parameters=None,
                                                  draw_nuisance_toys=None, normalize=False,
                                                  log=False, observable_labels=None, n_bins=50,
                                                  line_labels=None, colors=None, linestyle=None,
                                                  linewidths=1.5, toy_linewidths=0.5, alpha=0.15,
                                                  toy_alpha=0.75, n_events=None, n_toys=100,
                                                  n_cols=3, quantiles_for_range=(0.025, 0.975),
                                                  sample_only_from_closest_benchmark=True)
```

Plots one-dimensional histograms of observables in a MadMiner file for a given set of benchmarks.

#### Parameters

##### **filename**

[str] Filename of a MadMiner HDF5 file.

##### **observables**

[list of str or None, optional] Which observables to plot, given by a list of their names. If None, all observables in the file are plotted. Default value: None.

##### **parameter\_points**

[list of (str or ndarray) or None, optional] Which parameter points to use for histogramming the data. Given by a list, each element can either be the name of a benchmark in the MadMiner file, or an ndarray specifying any parameter point in a morphing setup. If None, all physics (non-nuisance) benchmarks defined in the MadMiner file are plotted. Default value: None.

##### **uncertainties**

[{"nuisance", "none"}, optional] Defines how uncertainty bands are drawn. With "nuisance", the variation in cross section from all nuisance parameters is added in quadrature. With "none", no error bands are drawn.

##### **nuisance\_parameters**

[None or list of int, optional] If uncertainties is "nuisance", this can restrict which nuisance parameters are used to draw the uncertainty bands. Each entry of this list is the index of one nuisance parameter (same order as in the MadMiner file).

**draw\_nuisance\_toys**

[None or int, optional] If not None and uncertainties is “nuisance”, sets the number of nuisance toy distributions that are drawn (in addition to the error bands).

**normalize**

[bool, optional] Whether the distribution is normalized to the total cross section. Default value: False.

**log**

[bool, optional] Whether to draw the y axes on a logarithmic scale. Default value: False.

**observable\_labels**

[None or list of (str or None), optional] x-axis labels naming the observables. If None, the observable names from the MadMiner file are used. Default value: None.

**n\_bins**

[int, optional] Number of histogram bins. Default value: 50.

**line\_labels**

[None or list of (str or None), optional] Labels for the different parameter points. If None and if parameter\_points is None, the benchmark names from the MadMiner file are used. Default value: None.

**colors**

[None or str or list of str, optional] Matplotlib line (and error band) colors for the distributions. If None, uses default colors. Default value: None.

**linestyles**

[None or str or list of str, optional] Matplotlib line styles for the distributions. If None, uses default linestyles. Default value: None.

**linewidths**

[float or list of float, optional] Line widths for the contours. Default value: 1.5.

**toy\_linewidths**

[float or list of float or None, optional] Line widths for the toy replicas, if uncertainties is “nuisance” and draw\_nuisance\_toys is not None. If None, linewidths is used. Default value: 1.

**alpha**

[float, optional] alpha value for the uncertainty bands. Default value: 0.25.

**toy\_alpha**

[float, optional] alpha value for the toy replicas, if uncertainties is “nuisance” and draw\_nuisance\_toys is not None. Default value: 0.75.

**n\_events**

[None or int, optional] If not None, sets the number of events from the MadMiner file that will be analyzed and plotted. Default value: None.

**n\_toys**

[int, optional] Number of toy nuisance parameter vectors used to estimate the systematic uncertainties. Default value: 100.

**n\_cols**

[int, optional] Number of columns of subfigures in the plot. Default value: 3.

**quantiles\_for\_range**

[tuple of two float, optional] Tuple (*min\_quantile*, *max\_quantile*) that defines how the observable range is determined for each panel. Default: (0.025, 0.075).

**sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

**Returns****figure**

[Figure] Plot as Matplotlib Figure instance.

```
madminer.plotting.distributions.plot_histograms(histos, observed=None, observed_weights=None,
                                              xrange=None, yrange=None, zrange=None,
                                              log=False, histo_labels=None,
                                              observed_label='Data', xlabel=None, ylabel=None,
                                              zlabel=None, colors=None, linestyle=None,
                                              linewidths=1.5, markercolor='black',
                                              markersize=20.0, cmap='viridis', n_cols=2)
```

## 14.3 madminer.plotting.fisherinformation module

```
madminer.plotting.fisherinformation.plot_distribution_of_information(xbins, xsecs,
                                                                    fisher_information_matrices,
                                                                    fisher_information_matrices_aux=None,
                                                                    xlabel=None, xmin=None,
                                                                    xmax=None,
                                                                    log_xsec=False,
                                                                    norm_xsec=True,
                                                                    epsilon=1e-09,
                                                                    figsize=(5.4, 4.5),
                                                                    fontsize=None)
```

Plots the distribution of the cross section together with the distribution of the Fisher information.

**Parameters****xbins**

[list of float] Bin boundaries.

**xsecs**

[list of float] Cross sections (in pb) per bin.

**fisher\_information\_matrices**

[list of ndarray] Fisher information matrices for each bin.

**fisher\_information\_matrices\_aux**

[list of ndarray or None, optional] Additional Fisher information matrices for each bin (will be plotted with a dashed line).

**xlabel**

[str or None, optional] Label for the x axis.

**xmin**

[float or None, optional] Minimum value for the x axis.

**xmax**

[float or None, optional] Maximum value for the x axis.

**log\_xsec**

[bool, optional] Whether to plot the cross section on a logarithmic y axis.

**norm\_xsec**

[bool, optional] Whether the cross sections are normalized to 1.

**epsilon**

[float, optional] Numerical factor.

**figsize**

[tuple of float, optional] Figure size, default: (5.4, 4.5)

**fontsize: float, optional**

Fontsize, default None

## Returns

**figure**

[Figure] Plot as Matplotlib Figure instance.

```
madminer.plotting.fisherinformation.plot_fisher_information_contours_2d(fisher_information_matrices,
                                                                           fisher_information_covariances=None,
                                                                           reference_thetas=None,
                                                                           contour_distance=1.0,
                                                                           xlabel=r'$\theta_0$',
                                                                           ylabel=r'$\theta_1$',
                                                                           xrange=(-1.0, 1.0),
                                                                           yrange=(-1.0, 1.0),
                                                                           labels=None,
                                                                           inline_labels=None,
                                                                           resolution=500,
                                                                           colors=None,
                                                                           linestyle=None,
                                                                           linewidths=1.5,
                                                                           alphas=1.0,
                                                                           alphas_uncertainties=0.25,
                                                                           sigma_uncertainties=1,
                                                                           ax=None)
```

Visualizes 2x2 Fisher information matrices as contours of constant Fisher distance from a reference point  $\theta_0$ .

The local (tangent-space) approximation is used: distances  $d(\theta)$  are given by  $d(\theta)^2 = (\theta - \theta_0)_i I_{ij} (\theta - \theta_0)_j$ , summing over  $i$  and  $j$ .

## Parameters

**fisher\_information\_matrices**

[list of ndarray] Fisher information matrices, each with shape (2,2).

**fisher\_information\_covariances**

[None or list of (ndarray or None), optional] Covariance matrices for the Fisher information matrices. Has to have the same length as `fisher_information_matrices`, and each entry has to be None (no uncertainty) or a tensor with shape (2,2,2,2). Default value: None.

**reference\_thetas**

[None or list of (ndarray or None), optional] Reference points from which the distances are calculated. If None, the origin (0,0) is used. Default value: None.

**contour\_distance**

[float, optional.] Distance threshold at which the contours are drawn. Default value: 1.

**xlabel**

[str, optional] Label for the x axis. Default value:  $r'\theta_0$ .



**ylabel**

[str, optional] Label for the y axis. Default value:  $r' \$ \text{heta}_1 \$$ .

**xrange**

[tuple of float, optional] Range (*min*, *max*) for the x axis. Default value: (-1., 1.).

**yrange**

[tuple of float, optional] Range (*min*, *max*) for the y axis. Default value: (-1., 1.).

**labels**

[None or list of (str or None), optional] Legend labels for the contours. Default value: None.

**inline\_labels**

[None or list of (str or None), optional] Inline labels for the contours. Default value: None.

**resolution**

[int] Number of points per axis for the calculation of the distances. Default value: 500.

**colors**

[None or str or list of str, optional] Matplotlib line (and error band) colors for the contours. If None, uses default colors. Default value: None.

**linestyles**

[None or str or list of str, optional] Matplotlib line styles for the contours. If None, uses default linestyles. Default value: None.

**linewidths**

[float or list of float, optional] Line widths for the contours. Default value: 1.5.

**alphas**

[float or list of float, optional] Opacities for the contours. Default value: 1.

**alphas\_uncertainties**

[float or list of float, optional] Opacities for the error bands. Default value: 0.25.

**sigma\_uncertainties**

[float, optional] Number of gaussian sigmas used when presenting uncertainty bands. Default value: 1.

**ax: axes or None, optional**

Predefined axes as part of figure instead of standalone figure. Default: None

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

`madminer.plotting.fisherinformation.plot_fisherinfo_barplot(fisher_information_matrices, labels, determinant_indices=None, eigenvalue_colors=None, bar_colors=None)`

**Parameters**

**fisher\_information\_matrices**

[list of ndarray] Fisher information matrices

**labels**

[list of str] Labels for the x axis

**determinant\_indices**

[list of int or None, optional] If not None, the determinants will be based only on the indices given here. Default value: None.

**eigenvalue\_colors**

[None or list of str] Colors for the eigenvalue decomposition. If None, default colors are used. Default value: None.

**bar\_colors**

[None or list of str] Colors for the determinant bars. If None, default colors are used. Default value: None.

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

## 14.4 madminer.plotting.limits module

`madminer.plotting.limits.plot_pvalue_limits(p_values, best_fits, labels, grid_ranges, grid_resolutions, levels=[0.32], single_plot=True, show_index=None, xlabel='$\theta_0$', ylabel='$\theta_1$', p_val_min=0.001, p_val_max=1)`

Function that plots the limits obtained from the AsymptoticLimits, Likelihood, FisherInformation and Information Geometry class. Note that only 2 dimensional grids are supported.

**Parameters**

**p\_values**

[list of ndarray or dict] List/dictionary of p-values with shape (nmethods, ngridpoints)

**best\_fits**

[list of int or dict] List/dictionary of best fit points for each method with shape (nmethods)

**labels**

[list of string or None] List/dictionary of best labels for each method with shape (nmethods). If None, it is assumed that dictionaries are provided and all entries will be used.

**grid\_ranges**

[list of (tuple of float) or None, optional] Specifies the boundaries of the parameter grid on which the p-values are evaluated. It should be [(*min*, *max*), (*min*, *max*), ..., (*min*, *max*)], where the list goes over all parameters and *min* and *max* are float. If None, *thetas\_eval* has to be given. Default: None.

**grid\_resolutions**

[int or list of int, optional] Resolution of the parameter space grid on which the p-values are evaluated. If int, the resolution is the same along every dimension of the hypercube. If list of int, the individual entries specify the number of points along each parameter individually. Doesn't have any effect if *grid\_ranges* is None. Default value: 25.

**levels**

[list of float, optional] list of p-values used to draw contour lines. Default: [0.32]

**single\_plot**

[bool, optional] If True, only one summary plot is shown which contains confidence contours and best fit points for all methods, and the p-value grid for a selected method (if *show\_index* is not None). If False, additional plots with the p-value grid, confidence contours and best fit points for all methods are provided. Default: True

**show\_index**

[int, optional] If None, no p-value grid is shown in summary plot. If *show\_index*=*n*, the p-value grid of the *n*th method is shown in the summary plot. Default is None.

**xlabel,ylabel**

[string, optional] Labels for the x and y axis. Default: xlabel=r'\$\theta\_0\$' and ylabel=r'\$\theta\_1\$'.

**p\_val\_min,p\_val\_max**

[float, optional] Plot range for p-values. Default: p\_val\_min=0.001 and p\_val\_max=1.

## 14.5 madminer.plotting.morphing module

`madminer.plotting.morphing.plot_1d_morphing_basis(morpher, xlabel='$\theta$', xrange=(-1.0, 1.0), resolution=100)`

Visualizes a morphing basis and morphing errors for problems with a two-dimensional parameter space.

**Parameters**

**morpher**

[PhysicsMorpher] PhysicsMorpher instance with defined basis.

**xlabel**

[str, optional] Label for the x axis. Default value: r'\$\theta\$'.

**xrange**

[tuple of float, optional] Range (*min*, *max*) for the x axis. Default value: (-1., 1.).

**resolution**

[int, optional] Number of points per axis for the rendering of the squared morphing weights. Default value: 100.

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

`madminer.plotting.morphing.plot_2d_morphing_basis(morpher, xlabel='$\theta_0$', ylabel='$\theta_1$', xrange=(-1.0, 1.0), yrange=(-1.0, 1.0), crange=(1.0, 100.0), resolution=100)`

Visualizes a morphing basis and morphing errors for problems with a two-dimensional parameter space.

**Parameters**

**morpher**

[PhysicsMorpher] PhysicsMorpher instance with defined basis.

**xlabel**

[str, optional] Label for the x axis. Default value: r'\$\theta\_0\$'.

**ylabel**

[str, optional] Label for the y axis. Default value: r'\$\theta\_1\$'.

**xrange**

[tuple of float, optional] Range (*min*, *max*) for the x axis. Default value: (-1., 1.).

**yrange**

[tuple of float, optional] Range (*min*, *max*) for the y axis. Default value: (-1., 1.).

**crange**

[tuple of float, optional] Range (*min*, *max*) for the color map. Default value: (1., 100.).

**resolution**

[int, optional] Number of points per axis for the rendering of the squared morphing weights.  
Default value: 100.

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

`madminer.plotting.morphing.plot_nd_morphing_basis_scatter(morpher, crange=(1.0, 100.0),  
n_test_thetas=1000)`

Visualizes a morphing basis and morphing errors with scatter plots between each pair of operators.

**Parameters**

**morpher**

[PhysicsMorpher] PhysicsMorpher instance with defined basis.

**crange**

[tuple of float, optional] Range (*min*, *max*) for the color map. Default value: (1. 100.).

**n\_test\_thetas**

[int, optional] Number of random points evaluated. Default value: 1000.

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

`madminer.plotting.morphing.plot_nd_morphing_basis_slices(morpher, crange=(1.0, 100.0),  
resolution=50)`

Visualizes a morphing basis and morphing errors with two-dimensional slices through parameter space.

**Parameters**

**morpher**

[PhysicsMorpher] PhysicsMorpher instance with defined basis.

**crange**

[tuple of float, optional] Range (*min*, *max*) for the color map.

**resolution**

[int, optional] Number of points per panel and axis for the rendering of the squared morphing weights. Default value: 50.

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

## 14.6 madminer.plotting.uncertainties module

`madminer.plotting.uncertainties.plot_systematics(filename, theta, observable, obs_label, obs_range,  
n_bins=50, n_events=None, n_toys=100,  
linecolor='black', bandcolors=None,  
band_alpha=0.2, ratio_range=(0.8, 1.2))`

Plots absolute and relative uncertainty bands for all systematic uncertainties in a histogram of one observable in a MadMiner file.

**Parameters**

**filename**

[str] Filename of a MadMiner HDF5 file.

**theta**

[ndarray, optional] Which parameter points to use for histogramming the data.

**observable**

[str] Which observable to plot, given by its name in the MadMiner file.

**obs\_label**

[str] x-axis label naming the observable.

**obs\_range**

[tuple of two float] Range to be plotted for the observable.

**n\_bins**

[int] Number of bins. Default value: 50.

**n\_events**

[None or int, optional] If not None, sets the number of events from the MadMiner file that will be analyzed and plotted. Default value: None.

**n\_toys**

[int, optional] Number of toy nuisance parameter vectors used to estimate the systematic uncertainties. Default value: 100.

**linecolor**

[str, optional] Line color for central prediction. Default value: "black".

**bandcolors**

[None or list of str, optional] Error band colors. Default value: None.

**ratio\_range**

[tuple of two float] y-axis range for the plots of the ratio to the central prediction. Default value: (0.8, 1.2).

**Returns****figure**

[Figure] Plot as Matplotlib Figure instance.

```
madminer.plotting.uncertainties.plot_uncertainty(filename, theta, observable, obs_label, obs_range,
                                                  n_bins=50, systematics=None, n_events=None,
                                                  n_toys=100, linecolor='black',
                                                  bandcolor1='#CC002E', bandcolor2='orange',
                                                  ratio_range=(0.8, 1.2))
```

Plots absolute and relative uncertainty bands in a histogram of one observable in a MadMiner file.

**Parameters****filename**

[str] Filename of a MadMiner HDF5 file.

**theta**

[ndarray, optional] Which parameter points to use for histogramming the data.

**observable**

[str] Which observable to plot, given by its name in the MadMiner file.

**obs\_label**

[str] x-axis label naming the observable.

**obs\_range**

[tuple of two float] Range to be plotted for the observable.

**n\_bins**

[int] Number of bins. Default value: 50.

**systematics**

[None or list of str, optional] This can restrict which nuisance parameters are used to draw the uncertainty bands. Each entry of this list is the name of a systematic uncertainty (see *MadMiner.add\_systematics()*).

**n\_events**

[None or int, optional] If not None, sets the number of events from the MadMiner file that will be analyzed and plotted. Default value: None.

**n\_toys**

[int, optional] Number of toy nuisance parameter vectors used to estimate the systematic uncertainties. Default value: 100.

**linecolor**

[str, optional] Line color for central prediction. Default value: "black".

**bandcolor1**

[str, optional] Error band color for 1 sigma uncertainty. Default value: "#CC002E".

**bandcolor2**

[str, optional] Error band color for 2 sigma uncertainty. Default value: "orange".

**ratio\_range**

[tuple of two float] y-axis range for the plots of the ratio to the central prediction. Default value: (0.8, 1.2).

**Returns**

**figure**

[Figure] Plot as Matplotlib Figure instance.

## 14.7 Module contents

## MADMINER.SAMPLING PACKAGE

### 15.1 Submodules

### 15.2 `madminer.sampling.combine` module

```
madminer.sampling.combine.combine_and_shuffle(input_filenames: List[str], output_filename: str,  
                                              k_factors: Optional[Union[List[float], float]] = None,  
                                              recalculate_header: bool = True)
```

Combines multiple MadMiner files into one, and shuffles the order of the events.

Note that this function assumes that all samples are generated with the same setup, including identical benchmarks (and thus morphing setup). If it is used with samples with different settings, there will be wrong results! There are no explicit cross checks in place yet!

#### Parameters

##### **input\_filenames**

[list of str] List of paths to the input MadMiner files.

##### **output\_filename**

[str] Path to the combined MadMiner file.

##### **k\_factors**

[float or list of float, optional] Multiplies the weights in `input_filenames` with a universal factor (if `k_factors` is a float) or with independent factors (if it is a list of float). Default value: `None`.

##### **recalculate\_header**

[bool, optional] Recalculates the total number of events. Default value: `True`.

#### Returns

`None`

## 15.3 madminer.sampling.parameters module

`madminer.sampling.parameters.benchmark(benchmark_name)`

Utility function to be used as input to various SampleAugmenter functions, specifying a single parameter benchmark.

### Parameters

#### **benchmark\_name**

[str] Name of the benchmark (as in *madminer.core.MadMiner.add\_benchmark*)

### Returns

#### **output**

[tuple] Input to various SampleAugmenter functions

`madminer.sampling.parameters.benchmarks(benchmark_names)`

Utility function to be used as input to various SampleAugmenter functions, specifying multiple parameter benchmarks.

### Parameters

#### **benchmark\_names**

[list of str] List of names of the benchmarks (as in *madminer.core.MadMiner.add\_benchmark*)

### Returns

#### **output**

[tuple] Input to various SampleAugmenter functions

`madminer.sampling.parameters.iid_nuisance_parameters(shape='gaussian', param0=0.0, param1=1.0)`

Utility function to be used as input to various SampleAugmenter functions, specifying that nuisance parameters are fixed at their nominal values.

### Parameters

#### **shape**

[["flat", "gaussian"], optional] Parameter prior shape. Default value: "gaussian".

#### **param0**

[float, optional] Gaussian mean or flat lower bound. Default value: 0.0.

#### **param1**

[float, optional] Gaussian std or flat upper bound. Default value: 1.0.

### Returns

#### **output**

[tuple] Input to various SampleAugmenter functions.

`madminer.sampling.parameters.morphing_point(theta)`

Utility function to be used as input to various SampleAugmenter functions, specifying a single parameter point theta in a morphing setup.

### Parameters

#### **theta**

[ndarray or list] Parameter point with shape (*n\_parameters*,)

### Returns



**output**

[tuple] Input to various SampleAugmenter functions

`madminer.sampling.parameters.morphing_points(thetas)`

Utility function to be used as input to various SampleAugmenter functions, specifying multiple parameter points theta in a morphing setup.

**Parameters****thetas**

[ndarray or list of lists or list of ndarrays] Parameter points with shape (*n\_thetas*, *n\_parameters*)

**Returns****output**

[tuple] Input to various SampleAugmenter functions

`madminer.sampling.parameters.nominal_nuisance_parameters()`

Utility function to be used as input to various SampleAugmenter functions, specifying that nuisance parameters are fixed at their nominal values.

**Returns****output**

[tuple] Input to various SampleAugmenter functions

`madminer.sampling.parameters.random_morphing_points(n_thetas, priors)`

Utility function to be used as input to various SampleAugmenter functions, specifying random parameter points sampled from a prior in a morphing setup.

**Parameters****n\_thetas**

[int] Number of parameter points to be sampled

**priors**

[list of tuples] Priors for each parameter is characterized by a tuple of the form (*prior\_shape*, *prior\_param\_0*, *prior\_param\_1*). Currently, the supported *prior\_shapes* are *flat*, in which case the two other parameters are the lower and upper bound of the flat prior, and *gaussian*, in which case they are the mean and standard deviation of a Gaussian.

**Returns****output**

[tuple] Input to various SampleAugmenter functions

## 15.4 madminer.sampling.sampleaugmenter module

`class madminer.sampling.sampleaugmenter.SampleAugmenter(filename, disable_morphing=False, include_nuisance_parameters=True)`

Bases: [\*DataAnalyzer\*](#)

Sampling / unweighting and data augmentation.

After the generated events have been analyzed and the observables and weights have been saved into a MadMiner file, for instance with `madminer.delphes.DelphesReader` or `madminer.lhe.LHEReader`, the next step is typically the generation of training and evaluation data for the machine learning algorithms. This generally involves two

(related) tasks: unweighting, i.e. the creation of samples that do not carry individual weights but follow some distribution, and the extraction of the joint likelihood ratio and / or joint score (the “augmented data”).

After initializing *SampleAugmenter* with the filename of a MadMiner file, this is done with a single function call. Depending on the downstream inference algorithm, there are different possibilities:

- *SampleAugmenter.sample\_train\_plain()* creates plain training samples without augmented data.
- *SampleAugmenter.sample\_train\_local()* creates training samples for local methods based on the score, such as SALLY and SALLINO.
- *SampleAugmenter.sample\_train\_density()* creates training samples for non-local methods based on density estimation and the score, such as SCANDAL.
- *SampleAugmenter.sample\_train\_ratio()* creates training samples for non-local, ratio-based methods like RASCAL or ALICE.
- *SampleAugmenter.sample\_train\_more\_ratios()* does the same, but can extract joint ratios and scores at more parameter points. This additional information can be used efficiently in the setup with a “doubly parameterized” likelihood ratio estimator that models the dependence on both the numerator and denominator hypothesis.
- *SampleAugmenter.sample\_test()* creates evaluation samples for all methods.

Please see the tutorial for a walkthrough.

For the curious, let us explain these steps in a little bit more detail (assuming a morphing setup):

- The sample augmentation step starts from a set of events  $(x_i, z_i)$  together with corresponding weights for each morphing basis point  $\theta_b, p(x_i, z_i | \theta_b)$ .
- Morphing: Assume we want to generate data sampled from a parameter point  $\theta$ , which is not necessarily one of the basis points  $\theta_b$ . Using the morphing structure, the event weights for  $p(x_i, z_i | \theta)$  can be calculated. Note that the events (phase-space points)  $(x_i, z_i)$  are not changed, only their weights.
- Unweighting: For the machine learning part, such a weighted event sample is not practical. Instead we aim for an unweighted one, in which events can appear multiple times. If the user request  $N$  events (which can be larger than the original number of events in the MadGraph runs), *SampleAugmenter* will draw  $N$  samples  $(x_i, z_i)$  from the discrete distribution  $p(x_i, z_i | \theta)$ . In other words, it draws (with replacement)  $N$  of the original events from MadGraph, with probabilities given by the morphing setup before. This is similar to what *np.random.choice()* does.
- Augmentation: For each of the drawn samples, the morphing setup can be used to calculate the joint likelihood ratio and / or the joint score (this depends on which *SampleAugmenter* function is called).

## Parameters

### filename

[str] Path to MadMiner file (for instance the output of *madminer.delphe.DelphesProcessor.save()*).

### disable\_morphing

[bool, optional] If True, the morphing setup is not loaded from the file. Default value: False.

### include\_nuisance\_parameters

[bool, optional] If True, nuisance parameters are taken into account. Default value: True.

## Methods

<code>cross_sections(theta[, nu])</code>	Calculates the total cross sections for all specified thetas.
<code>event_loader([start, end, batch_size, ...])</code>	Yields batches of events in the MadMiner file.
<code>sample_test(theta, n_samples[, nu, ...])</code>	Extracts evaluation samples $x \sim p(x theta)$ without any augmented data.
<code>sample_train_density(theta, n_samples[, nu, ...])</code>	Extracts training samples $x \sim p(x theta)$ as well as the joint score $t(x, z theta)$ , where theta is sampled from a prior.
<code>sample_train_local(theta, n_samples[, nu, ...])</code>	Extracts training samples $x \sim p(x theta)$ as well as the joint score $t(x, z theta)$ .
<code>sample_train_more_ratios(theta0, theta1, ...)</code>	Extracts training samples $x \sim p(x theta0)$ and $x \sim p(x theta1)$ together with the class label $y$ , the joint likelihood ratio $r(x, z theta0, theta1)$ , and the joint score $t(x, z theta0)$ .
<code>sample_train_plain(theta, n_samples[, nu, ...])</code>	Extracts plain training samples $x \sim p(x theta)$ without any augmented data.
<code>sample_train_ratio(theta0, theta1, n_samples)</code>	Extracts training samples $x \sim p(x theta0)$ and $x \sim p(x theta1)$ together with the class label $y$ , the joint likelihood ratio $r(x, z theta0, theta1)$ , and, if morphing is set up, the joint score $t(x, z theta0)$ .
<code>weighted_events([theta, nu, start_event, ...])</code>	Returns all events together with the benchmark weights (if theta is None) or weights for a given theta.
<code>xsec_gradients(thetas[, nus, partition, ...])</code>	Returns the gradient of total cross sections with respect to parameters.
<code>xsecs([thetas, nus, partition, test_split, ...])</code>	Returns the total cross sections for benchmarks or parameter points.

### `cross_sections(theta, nu=None)`

Calculates the total cross sections for all specified thetas.

#### Parameters

##### **theta**

[tuple] Tuple (type, value) that defines the parameter point or prior over parameter points at which the cross section is calculated. Pass the output of the functions `benchmark()`, `benchmarks()`, `morphing_point()`, `morphing_points()`, or `random_morphing_points()`.

##### **nu**

[tuple or None, optional] Tuple (type, value) that defines the nuisance parameter point or prior over nuisance parameter points at which the cross section is calculated. Pass the output of the functions `benchmark()`, `benchmarks()`, `morphing_point()`, `morphing_points()`, or `random_morphing_points()`. Default value: None.

#### Returns

##### **thetas**

[ndarray] Parameter points with shape  $(n\_thetas, n\_parameters)$  or  $(n\_thetas, n\_parameters + n\_nuisance\_parameters)$ .

##### **xsecs**

[ndarray] Total cross sections in pb with shape  $(n\_thetas, )$ .

##### **xsec\_uncertainties**

[ndarray] Statistical uncertainties on the total cross sections in pb with shape  $(n\_thetas, )$ .

```
sample_test(theta, n_samples, nu=None, sample_only_from_closest_benchmark=True, folder=None,
             filename=None, test_split=0.2, validation_split=0.2, partition='test', n_processes=1,
             n_eff_forced=None, double_precision=False)
```

Extracts evaluation samples  $x \sim p(x|\theta)$  without any augmented data.

#### Parameters

##### **theta**

[tuple] Tuple (type, value) that defines the parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

##### **n\_samples**

[int] Total number of events to be drawn.

##### **nu**

[None or tuple, optional] Tuple (type, value) that defines the nuisance parameter point or prior over parameter points for the sampling. Default value: None

##### **sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

##### **folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npy format). Default value: None.

##### **filename**

[str or None] Filenames for the resulting samples. A prefix such as 'x' or 'theta0' as well as the extension '.npy' will be added automatically. Default value: None.

##### **test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

##### **validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

##### **partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default value: "test".

##### **n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, *n\_workers* sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

##### **n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n\_eff\_forced$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

##### **double\_precision**

[bool, optional] Use double floating-point precision. Default value: False

#### Returns

**x**

[ndarray] Observables with shape  $(n\_samples, n\_observables)$ . The same information is saved as a file in the given folder.

**theta**

[ndarray] Parameter points used for sampling with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where `event_probabilities` are the fractions of the cross section carried by each event.

**sample\_train\_density**(*theta*, *n\_samples*, *nu*=None, *sample\_only\_from\_closest\_benchmark*=True, *folder*=None, *filename*=None, *nuisance\_score*='auto', *test\_split*=0.2, *validation\_split*=0.2, *partition*='train', *n\_processes*=1, *n\_eff\_forced*=None, *double\_precision*=False)

Extracts training samples  $x \sim p(x|\theta)$  as well as the joint score  $t(x, z|\theta)$ , where  $\theta$  is sampled from a prior. This can be used for inference methods such as SCANDAL.

### Parameters

**theta**

[tuple] Tuple (type, value) that defines the numerator parameter point or prior over parameter points for the sampling. Pass the output of the functions `constant_benchmark_theta()`, `multiple_benchmark_thetas()`, `constant_morphing_theta()`, `multiple_morphing_thetas()`, or `random_morphing_thetas()`.

**n\_samples**

[int] Total number of events to be drawn.

**nu**

[None or tuple, optional] Tuple (type, value) that defines the nuisance parameter point or prior over parameter points for the sampling. Default value: None

**sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

**folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npy format). Default value: None.

**filename**

[str or None] Filenames for the resulting samples. A prefix such as 'x' or 'theta0' as well as the extension '.npy' will be added automatically. Default value: None.

**nuisance\_score**

[bool or "auto", optional] If True, the score with respect to the nuisance parameters (at the default position) will also be calculated. If False, only the score with respect to the physics parameters is calculated. For "auto", the nuisance score will be calculated if a nuisance setup is defined. Default: True.

**test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

**validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

**partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default value: "train".

**n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, `n_workers` sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

**n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n\_eff\_forced$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

**double\_precision**

[bool, optional] Use double floating-point precision. Default value: False.

**Returns**

**x**

[ndarray] Observables with shape  $(n\_samples, n\_observables)$ . The same information is saved as a file in the given folder.

**theta**

[ndarray] Parameter points used for sampling (and evaluation of the joint score) with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**t\_xz**

[ndarray] Joint score evaluated at theta with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where `event_probabilities` are the fractions of the cross section carried by each event.

**sample\_train\_local**(*theta*, *n\_samples*, *nu*=None, *sample\_only\_from\_closest\_benchmark*=True, *folder*=None, *filename*=None, *nuisance\_score*='auto', *test\_split*=0.2, *validation\_split*=0.2, *partition*='train', *n\_processes*=1, *log\_message*=True, *n\_eff\_forced*=None, *double\_precision*=False)

Extracts training samples  $x \sim p(x|\theta)$  as well as the joint score  $t(x, z|\theta)$ . This can be used for inference methods such as SALLY and SALLINO.

**Parameters**

**theta**

[tuple] Tuple (type, value) that defines the parameter point for the sampling. This is also where the score is evaluated. Pass the output of the functions `constant_benchmark_theta()` or `constant_morphing_theta()`.

**n\_samples**

[int] Total number of events to be drawn.

**nu**

[None or tuple, optional] Tuple (type, value) that defines the nuisance parameter point or prior over parameter points for the sampling. Default value: None

**sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

**folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npy format). Default value: None.

**filename**

[str or None] Filenames for the resulting samples. A prefix such as 'x' or 'theta0' as well as the extension '.npy' will be added automatically. Default value: None.

**nuisance\_score**

[bool or "auto", optional] If True, the score with respect to the nuisance parameters (at the default position) will also be calculated. If False, only the score with respect to the physics parameters is calculated. For "auto", the nuisance score will be calculated if a nuisance setup is defined. Default: True.

**test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

**validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

**partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default value: "train".

**n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, `n_workers` sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

**log\_message**

[bool, optional] If True, logging output. This option is only designed for internal use.

**n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n\_eff\_forced$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

**double\_precision**

[bool, optional] Use double floating-point precision. Default value: False.

**Returns**

**x**

[ndarray] Observables with shape  $(n\_samples, n\_observables)$ . The same information is saved as a file in the given folder.

**theta**

[ndarray] Parameter points used for sampling (and evaluation of the joint score) with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**t\_xz**

[ndarray] Joint score evaluated at theta with shape  $(n\_samples, n\_parameters + n\_nuisance\_parameters)$  (if `nuisance_score` is True) or  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where `event_probabilities` are the fractions of the cross section carried by each event.

```
sample_train_more_ratios(theta0, theta1, n_samples, nu0=None, nu1=None,  
                           sample_only_from_closest_benchmark=True, folder=None, filename=None,  
                           additional_thetas=None, nuisance_score='auto', test_split=0.2,  
                           validation_split=0.2, partition='train', n_processes=1, n_eff_forced=None,  
                           double_precision=False)
```

Extracts training samples  $x \sim p(x|\theta_0)$  and  $x \sim p(x|\theta_1)$  together with the class label  $y$ , the joint likelihood ratio  $r(x, z|\theta_0, \theta_1)$ , and the joint score  $t(x, z|\theta_0)$ . This information can be used in inference methods such as CARL, ROLR, CASCAL, and RASCAL.

With the keyword *additional\_thetas*, this function allows to extract joint ratios and scores at more parameter points than just *theta0* and *theta1*. This additional information can be used efficiently in the setup with a “doubly parameterized” likelihood ratio estimator that models the dependence on both the numerator and denominator hypothesis.

### Parameters

#### **theta0**

Tuple (type, value) that defines the numerator parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

#### **theta1**

Tuple (type, value) that defines the denominator parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

#### **n\_samples**

[int] Total number of events to be drawn.

#### **nu0**

[None or tuple, optional] Tuple (type, value) that defines the numerator nuisance parameter point or prior over parameter points for the sampling. Default value: None

#### **nu1**

[None or tuple, optional] Tuple (type, value) that defines the denominator nuisance parameter point or prior over parameter points for the sampling. Default value: None

#### **sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

#### **folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npy format). Default value: None.

#### **filename**

[str or None] Filenames for the resulting samples. A prefix such as ‘x’ or ‘theta0’ as well as the extension ‘.npy’ will be added automatically. Default value: None.

#### **additional\_thetas**

[list of tuple or None] list of tuples (type, value) that defines additional theta points at which ratio and score are evaluated, and which are then used to create additional training data points. These can be efficiently used only in the “doubly parameterized” setup where a likelihood ratio estimator models the dependence of the likelihood ratio on both the numerator and denominator hypothesis. Pass the output of the helper functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*. Default value: None.



**nuisance\_score**

[bool or “auto”, optional] If True, the score with respect to the nuisance parameters (at the default position) will also be calculated. If False, only the score with respect to the physics parameters is calculated. For “auto”, the nuisance score will be calculated if a nuisance setup is defined. Default: True.

**test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

**validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

**partition**

[{“train”, “test”, “validation”, “all”}, optional] Which event partition to use. Default value: “train”.

**n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, `n_workers` sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

**n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n_{\text{eff\_forced}}$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

**double\_precision**

[bool, optional] Use double floating-point precision. Default value: False

**Returns**

**x**

[ndarray] Observables with shape  $(n_{\text{samples}}, n_{\text{observables}})$ . The same information is saved as a file in the given folder.

**theta0**

[ndarray] Numerator parameter points with shape  $(n_{\text{samples}}, n_{\text{parameters}})$ . The same information is saved as a file in the given folder.

**theta1**

[ndarray] Denominator parameter points with shape  $(n_{\text{samples}}, n_{\text{parameters}})$ . The same information is saved as a file in the given folder.

**y**

[ndarray] Class label with shape  $(n_{\text{samples}}, n_{\text{parameters}})$ .  $y=0$  (1) for events sample from the numerator (denominator) hypothesis. The same information is saved as a file in the given folder.

**r\_xz**

[ndarray] Joint likelihood ratio with shape  $(n_{\text{samples}},)$ . The same information is saved as a file in the given folder.

**t\_xz**

[ndarray] Joint score evaluated at theta0 with shape  $(n_{\text{samples}}, n_{\text{parameters}})$ . The same information is saved as a file in the given folder.

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where `event_probabilities` are the fractions of the cross section carried by each event.

```
sample_train_plain(theta, n_samples, nu=None, sample_only_from_closest_benchmark=True,  
                    folder=None, filename=None, test_split=0.2, validation_split=0.2, partition='train',  
                    n_processes=1, n_eff_forced=None, double_precision=False)
```

Extracts plain training samples  $x \sim p(x|\theta)$  without any augmented data. This can be use for standard inference methods such as ABC, histograms of observables, or neural density estimation techniques. It can also be used to create validation or calibration samples.

#### Parameters

##### **theta**

[tuple] Tuple (type, value) that defines the parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

##### **n\_samples**

[int] Total number of events to be drawn.

##### **nu**

[None or tuple, optional] Tuple (type, value) that defines the nuisance parameter point or prior over parameter points for the sampling. Default value: None

##### **sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

##### **folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npz format). Default value: None.

##### **filename**

[str or None] Filenames for the resulting samples. A prefix such as 'x' or 'theta0' as well as the extension '.npz' will be added automatically. Default value: None.

##### **test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

##### **validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

##### **partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default value: "train".

##### **n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, n\_workers sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

##### **n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n_{\text{eff\_forced}}$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

##### **double\_precision**

[bool, optional] Use double floating-point precision. Default value: False.

#### Returns

**x**

[ndarray] Observables with shape  $(n\_samples, n\_observables)$ . The same information is saved as a file in the given folder.

**theta**

[ndarray] Parameter points used for sampling with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where  $\text{event\_probabilities}$  are the fractions of the cross section carried by each event.

**sample\_train\_ratio**(*theta0, theta1, n\_samples, nu0=None, nu1=None, sample\_only\_from\_closest\_benchmark=True, folder=None, filename=None, nuisance\_score='auto', test\_split=0.2, validation\_split=0.2, partition='train', n\_processes=1, return\_individual\_n\_effective=False, n\_eff\_forced=None, double\_precision=False*)

Extracts training samples  $x \sim p(x|\theta_0)$  and  $x \sim p(x|\theta_1)$  together with the class label  $y$ , the joint likelihood ratio  $r(x, z|\theta_0, \theta_1)$ , and, if morphing is set up, the joint score  $t(x, z|\theta_0)$ . This information can be used in inference methods such as CARL, ROLR, CASCAL, and RASCAL.

### Parameters

**theta0**

[tuple] Tuple (type, value) that defines the numerator parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

**theta1**

[tuple] Tuple (type, value) that defines the denominator parameter point or prior over parameter points for the sampling. Pass the output of the functions *constant\_benchmark\_theta()*, *multiple\_benchmark\_thetas()*, *constant\_morphing\_theta()*, *multiple\_morphing\_thetas()*, or *random\_morphing\_thetas()*.

**n\_samples**

[int] Total number of events to be drawn.

**nu0**

[None or tuple, optional] Tuple (type, value) that defines the numerator nuisance parameter point or prior over parameter points for the sampling. Default value: None

**nu1**

[None or tuple, optional] Tuple (type, value) that defines the denominator nuisance parameter point or prior over parameter points for the sampling. Default value: None

**sample\_only\_from\_closest\_benchmark**

[bool, optional] If True, only weighted events originally generated from the closest benchmarks are used. Default value: True.

**folder**

[str or None] Path to the folder where the resulting samples should be saved (ndarrays in .npy format). Default value: None.

**filename**

[str or None] Filenames for the resulting samples. A prefix such as 'x' or 'theta0' as well as the extension '.npy' will be added automatically. Default value: None.

**nuisance\_score**

[bool or "auto", optional] If True, the score with respect to the nuisance parameters (at the

default position) will also be calculated. If False, only the score with respect to the physics parameters is calculated. For “auto”, the nuisance score will be calculated if a nuisance setup is defined. Default: True.

**test\_split**

[float or None, optional] Fraction of events reserved for the evaluation sample (that will not be used for any training samples). Default value: 0.2.

**validation\_split**

[float or None, optional] Fraction of events reserved for testing. Default value: 0.2.

**partition**

[{"train", "test", "validation", "all"}, optional] Which event partition to use. Default value: “train”.

**n\_processes**

[None or int, optional] If None or larger than 1, MadMiner will use multiprocessing to parallelize the sampling. In this case, `n_workers` sets the number of jobs running in parallel, and None will use the number of CPUs. Default value: 1.

**return\_individual\_n\_effective**

[bool, optional] Returns number of effective samples for each set individually. Default value: False.

**n\_eff\_forced**

[float, optional] If not None, MadMiner will require the relative weights of the events to be smaller than  $1/n\_eff\_forced$  and ignore other events. This can help to reduce statistical effects caused by a small number of events with very large weights obtained by the morphing procedure. Default value: None

**double\_precision**

[bool, optional] Use double floating-point precision. Default value: False

**Returns**

**x**

[ndarray] Observables with shape  $(n\_samples, n\_observables)$ . The same information is saved as a file in the given folder.

**theta0**

[ndarray] Numerator parameter points with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**theta1**

[ndarray] Denominator parameter points with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder.

**y**

[ndarray] Class label with shape  $(n\_samples, n\_parameters)$ .  $y=0$  (1) for events sample from the numerator (denominator) hypothesis. The same information is saved as a file in the given folder.

**r\_xz**

[ndarray] Joint likelihood ratio with shape  $(n\_samples,)$ . The same information is saved as a file in the given folder.

**t\_xz**

[ndarray or None] If morphing is set up, the joint score evaluated at theta0 with shape  $(n\_samples, n\_parameters)$ . The same information is saved as a file in the given folder. If morphing is not set up, None is returned (and no file is saved).

**effective\_n\_samples**

[int] Effective number of samples, defined as  $1/\max(\text{event\_probabilities})$ , where `event_probabilities` are the fractions of the cross section carried by each event.

## 15.5 Module contents



## INDICES AND TABLES

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





## PYTHON MODULE INDEX

### m

- `madminer.analysis`, 18
- `madminer.analysis.dataanalyzer`, 15
- `madminer.core`, 28
- `madminer.core.madminer`, 19
- `madminer.delphes`, 35
- `madminer.delphes.delphes_reader`, 29
- `madminer.fisherinformation`, 54
- `madminer.fisherinformation.geometry`, 37
- `madminer.fisherinformation.information`, 40
- `madminer.fisherinformation.manipulate`, 53
- `madminer.lhe`, 61
- `madminer.lhe.lhe_reader`, 55
- `madminer.likelihood`, 70
- `madminer.likelihood.base`, 63
- `madminer.likelihood.histo`, 63
- `madminer.likelihood.manipulate`, 66
- `madminer.likelihood.neural`, 69
- `madminer.limits`, 79
- `madminer.limits.asymptotic_limits`, 71
- `madminer.ml`, 112
- `madminer.ml.base`, 81
- `madminer.ml.double_parameterized_ratio`, 84
- `madminer.ml.ensemble`, 89
- `madminer.ml.likelihood`, 94
- `madminer.ml.lookup`, 99
- `madminer.ml.morphing_aware`, 99
- `madminer.ml.parameterized_ratio`, 104
- `madminer.ml.score`, 109
- `madminer.plotting`, 122
- `madminer.plotting.distributions`, 113
- `madminer.plotting.fisherinformation`, 115
- `madminer.plotting.limits`, 118
- `madminer.plotting.morphing`, 119
- `madminer.plotting.uncertainties`, 120
- `madminer.sampling`, 137
- `madminer.sampling.combine`, 123
- `madminer.sampling.parameters`, 124
- `madminer.sampling.sampleaugmenter`, 125



## INDEX

### A

`add_benchmark()` (*madminer.core.madminer.MadMiner* method), 20  
`add_cut()` (*madminer.delphes.delphes\_reader.DelphesReader* method), 30  
`add_cut()` (*madminer.lhe.lhe\_reader.LHEReader* method), 56  
`add_default_observables()` (*madminer.delphes.delphes\_reader.DelphesReader* method), 30  
`add_default_observables()` (*madminer.lhe.lhe\_reader.LHEReader* method), 56  
`add_efficiency()` (*madminer.lhe.lhe\_reader.LHEReader* method), 57  
`add_estimator()` (*madminer.ml.ensemble.Ensemble* method), 90  
`add_observable()` (*madminer.delphes.delphes\_reader.DelphesReader* method), 31  
`add_observable()` (*madminer.lhe.lhe\_reader.LHEReader* method), 57  
`add_observable_from_function()` (*madminer.delphes.delphes\_reader.DelphesReader* method), 32  
`add_observable_from_function()` (*madminer.lhe.lhe\_reader.LHEReader* method), 58  
`add_parameter()` (*madminer.core.madminer.MadMiner* method), 20  
`add_sample()` (*madminer.delphes.delphes\_reader.DelphesReader* method), 32  
`add_sample()` (*madminer.lhe.lhe\_reader.LHEReader* method), 58  
`add_systematics()` (*madminer.core.madminer.MadMiner* method), 21  
`analyse_delphes_samples()` (*mad-*

*miner.delphes.delphes\_reader.DelphesReader* method), 33  
`analyse_samples()` (*madminer.lhe.lhe\_reader.LHEReader* method), 59  
`asymptotic_p_value()` (*madminer.limits.asymptotic\_limits.AsymptoticLimits* method), 72  
`AsymptoticLimits` (class in *madminer.limits.asymptotic\_limits*), 71

### B

`BaseLikelihood` (class in *madminer.likelihood.base*), 63  
`benchmark()` (in module *madminer.sampling.parameters*), 124  
`benchmarks()` (in module *madminer.sampling.parameters*), 124

### C

`calculate_fisher_information()` (*madminer.ml.base.Estimator* method), 83  
`calculate_fisher_information()` (*madminer.ml.double\_parameterized\_ratio.DoubleParameterizedRatio* method), 85  
`calculate_fisher_information()` (*madminer.ml.ensemble.Ensemble* method), 90  
`calculate_fisher_information()` (*madminer.ml.likelihood.LikelihoodEstimator* method), 95  
`calculate_fisher_information_full_detector()` (*madminer.fisherinformation.information.FisherInformation* method), 41  
`calculate_fisher_information_full_truth()` (*madminer.fisherinformation.information.FisherInformation* method), 43  
`calculate_fisher_information_hist1d()` (*madminer.fisherinformation.information.FisherInformation* method), 43  
`calculate_fisher_information_hist2d()` (*madminer.fisherinformation.information.FisherInformation* method), 44

`calculate_fisher_information_nuisance_constraints()` (`madminer.fisherinformation.information.FisherInformation` method), 45  
`calculate_fisher_information_rate()` (`madminer.fisherinformation.information.FisherInformation` method), 45  
`combine_and_shuffle()` (in module `madminer.sampling.combine`), 123  
`ConditionalEstimator` (class in `madminer.ml.base`), 81  
`create_expected_negative_log_likelihood()` (`madminer.likelihood.base.BaseLikelihood` method), 63  
`create_expected_negative_log_likelihood()` (`madminer.likelihood.histo.HistoLikelihood` method), 64  
`create_expected_negative_log_likelihood()` (`madminer.likelihood.neural.NeuralLikelihood` method), 69  
`create_negative_log_likelihood()` (`madminer.likelihood.base.BaseLikelihood` method), 63  
`create_negative_log_likelihood()` (`madminer.likelihood.histo.HistoLikelihood` method), 65  
`create_negative_log_likelihood()` (`madminer.likelihood.neural.NeuralLikelihood` method), 69  
`cross_sections()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 127

## D

`DataAnalyzer` (class in `madminer.analysis.dataanalyzer`), 15  
`DelphesReader` (class in `madminer.delphes.delphes_reader`), 29  
`distance_contours()` (`madminer.fisherinformation.geometry.InformationGeometry` method), 37  
`DoubleParameterizedRatioEstimator` (class in `madminer.ml.double_parameterized_ratio`), 84

## E

`Ensemble` (class in `madminer.ml.ensemble`), 89  
`Estimator` (class in `madminer.ml.base`), 82  
`evaluate()` (`madminer.ml.base.Estimator` method), 83  
`evaluate()` (`madminer.ml.double_parameterized_ratio.DoubleParameterizedRatioEstimator` method), 86  
`evaluate()` (`madminer.ml.likelihood.LikelihoodEstimator` method), 96  
`evaluate()` (`madminer.ml.parameterized_ratio.ParameterizedRatioEstimator` method), 105  
`evaluate_log_likelihood()` (`madminer.ml.score.ScoreEstimator` method), 109  
`evaluate_log_likelihood()` (`madminer.ml.base.Estimator` method), 83  
`evaluate_log_likelihood()` (`madminer.ml.double_parameterized_ratio.DoubleParameterizedRatioEstimator` method), 86  
`evaluate_log_likelihood()` (`madminer.ml.ensemble.Ensemble` method), 91  
`evaluate_log_likelihood()` (`madminer.ml.likelihood.LikelihoodEstimator` method), 96  
`evaluate_log_likelihood()` (`madminer.ml.parameterized_ratio.ParameterizedRatioEstimator` method), 105  
`evaluate_log_likelihood()` (`madminer.ml.score.ScoreEstimator` method), 109  
`evaluate_log_likelihood_ratio()` (`madminer.ml.base.Estimator` method), 84  
`evaluate_log_likelihood_ratio()` (`madminer.ml.double_parameterized_ratio.DoubleParameterizedRatioEstimator` method), 86  
`evaluate_log_likelihood_ratio()` (`madminer.ml.ensemble.Ensemble` method), 92  
`evaluate_log_likelihood_ratio()` (`madminer.ml.likelihood.LikelihoodEstimator` method), 96  
`evaluate_log_likelihood_ratio()` (`madminer.ml.parameterized_ratio.ParameterizedRatioEstimator` method), 105  
`evaluate_log_likelihood_ratio_torch()` (`madminer.ml.parameterized_ratio.ParameterizedRatioEstimator` method), 106  
`evaluate_score()` (`madminer.ml.base.Estimator` method), 84  
`evaluate_score()` (`madminer.ml.double_parameterized_ratio.DoubleParameterizedRatioEstimator` method), 86  
`evaluate_score()` (`madminer.ml.ensemble.Ensemble` method), 92  
`evaluate_score()` (`madminer.ml.likelihood.LikelihoodEstimator` method), 97  
`evaluate_score()` (`madminer.ml.parameterized_ratio.ParameterizedRatioEstimator` method), 106  
`evaluate_score()` (`madminer.ml.score.ScoreEstimator` method), 109  
`event_loader()` (`madminer.analysis.dataanalyzer.DataAnalyzer` method), 144

method), 15  
 expected\_limits() (madminer.limits.asymptotic\_limits.AsymptoticLimits method), 72

## F

find\_trajectory() (madminer.fisherinformation.geometry.InformationGeometry method), 38  
 finite\_differences() (madminer.core.madminer.MadMiner method), 22  
 FisherInformation (class in madminer.fisherinformation.information), 40  
 fix\_params() (in module madminer.likelihood.manipulate), 66  
 full\_information() (madminer.fisherinformation.information.FisherInformation method), 46

## H

histo\_information() (madminer.fisherinformation.information.FisherInformation method), 47  
 histo\_information\_2d() (madminer.fisherinformation.information.FisherInformation method), 48  
 histogram\_of\_fisher\_information() (madminer.fisherinformation.information.FisherInformation method), 49  
 histogram\_of\_information() (madminer.fisherinformation.information.FisherInformation method), 50  
 histogram\_of\_sigma\_dsigma() (madminer.fisherinformation.information.FisherInformation method), 51  
 HistoLikelihood (class in madminer.likelihood.histo), 63

## I

iid\_nuisance\_parameters() (in module madminer.sampling.parameters), 124  
 information\_from\_formula() (madminer.fisherinformation.geometry.InformationGeometry method), 39  
 information\_from\_grid() (madminer.fisherinformation.geometry.InformationGeometry method), 39  
 InformationGeometry (class in madminer.fisherinformation.geometry), 37  
 initialize\_input\_transform() (madminer.ml.base.Estimator method), 84  
 initialize\_parameter\_transform() (madminer.ml.base.ConditionalEstimator method), 81

## L

LHEReader (class in madminer.lhe.lhe\_reader), 55  
 LikelihoodEstimator (class in madminer.ml.likelihood), 94  
 load() (madminer.core.madminer.MadMiner method), 22  
 load() (madminer.ml.base.ConditionalEstimator method), 81  
 load() (madminer.ml.base.Estimator method), 84  
 load() (madminer.ml.ensemble.Ensemble method), 93  
 load() (madminer.ml.score.ScoreEstimator method), 110  
 load\_estimator() (in module madminer.ml.lookup), 99

## M

MadMiner (class in madminer.core.madminer), 19  
 madminer.analysis  
   module, 18  
   madminer.analysis.dataanalyzer  
     module, 15  
   madminer.core  
     module, 28  
     madminer.core.madminer  
       module, 19  
   madminer.delphes  
     module, 35  
     madminer.delphes.delphes\_reader  
       module, 29  
   madminer.fisherinformation  
     module, 54  
     madminer.fisherinformation.geometry  
       module, 37  
     madminer.fisherinformation.information  
       module, 40  
     madminer.fisherinformation.manipulate  
       module, 53  
   madminer.lhe  
     module, 61  
     madminer.lhe.lhe\_reader  
       module, 55  
   madminer.likelihood  
     module, 70  
     madminer.likelihood.base  
       module, 63  
     madminer.likelihood.histo  
       module, 63  
     madminer.likelihood.manipulate  
       module, 66  
     madminer.likelihood.neural  
       module, 69

- madminer.limits
  - module, 79
- madminer.limits.asymptotic\_limits
  - module, 71
- madminer.ml
  - module, 112
- madminer.ml.base
  - module, 81
- madminer.ml.double\_parameterized\_ratio
  - module, 84
- madminer.ml.ensemble
  - module, 89
- madminer.ml.likelihood
  - module, 94
- madminer.ml.lookup
  - module, 99
- madminer.ml.morphing\_aware
  - module, 99
- madminer.ml.parameterized\_ratio
  - module, 104
- madminer.ml.score
  - module, 109
- madminer.plotting
  - module, 122
- madminer.plotting.distributions
  - module, 113
- madminer.plotting.fisherinformation
  - module, 115
- madminer.plotting.limits
  - module, 118
- madminer.plotting.morphing
  - module, 119
- madminer.plotting.uncertainties
  - module, 120
- madminer.sampling
  - module, 137
- madminer.sampling.combine
  - module, 123
- madminer.sampling.parameters
  - module, 124
- madminer.sampling.sampleaugmenter
  - module, 125
- module
  - madminer.analysis, 18
  - madminer.analysis.dataanalyzer, 15
  - madminer.core, 28
  - madminer.core.madminer, 19
  - madminer.delphes, 35
  - madminer.delphes.delphes\_reader, 29
  - madminer.fisherinformation, 54
  - madminer.fisherinformation.geometry, 37
  - madminer.fisherinformation.information, 40
  - madminer.fisherinformation.manipulate, 53

- madminer.lhe, 61
- madminer.lhe.lhe\_reader, 55
- madminer.likelihood, 70
- madminer.likelihood.base, 63
- madminer.likelihood.histo, 63
- madminer.likelihood.manipulate, 66
- madminer.likelihood.neural, 69
- madminer.limits, 79
- madminer.limits.asymptotic\_limits, 71
- madminer.ml, 112
- madminer.ml.base, 81
- madminer.ml.double\_parameterized\_ratio, 84
- madminer.ml.ensemble, 89
- madminer.ml.likelihood, 94
- madminer.ml.lookup, 99
- madminer.ml.morphing\_aware, 99
- madminer.ml.parameterized\_ratio, 104
- madminer.ml.score, 109
- madminer.plotting, 122
- madminer.plotting.distributions, 113
- madminer.plotting.fisherinformation, 115
- madminer.plotting.limits, 118
- madminer.plotting.morphing, 119
- madminer.plotting.uncertainties, 120
- madminer.sampling, 137
- madminer.sampling.combine, 123
- madminer.sampling.parameters, 124
- madminer.sampling.sampleaugmenter, 125
- morphing\_point() (in module madminer.sampling.parameters), 124
- morphing\_points() (in module madminer.sampling.parameters), 125
- MorphingAwareRatioEstimator (class in madminer.ml.morphing\_aware), 99

## N

- NeuralLikelihood (class in madminer.likelihood.neural), 69
- nominal\_nuisance\_parameters() (in module madminer.sampling.parameters), 125
- nuisance\_constraint\_information() (madminer.fisherinformation.information.FisherInformation method), 52

## O

- observed\_limits() (madminer.limits.asymptotic\_limits.AsymptoticLimits method), 75

## P

- ParameterizedRatioEstimator (class in madminer.ml.parameterized\_ratio), 104

- `plot_1d_morphing_basis()` (in module `madminer.plotting.morphing`), 119
- `plot_2d_morphing_basis()` (in module `madminer.plotting.morphing`), 119
- `plot_distribution_of_information()` (in module `madminer.plotting.fisherinformation`), 115
- `plot_distributions()` (in module `madminer.plotting.distributions`), 113
- `plot_fisher_information_contours_2d()` (in module `madminer.plotting.fisherinformation`), 116
- `plot_fisherinfo_barplot()` (in module `madminer.plotting.fisherinformation`), 117
- `plot_histograms()` (in module `madminer.plotting.distributions`), 115
- `plot_nd_morphing_basis_scatter()` (in module `madminer.plotting.morphing`), 120
- `plot_nd_morphing_basis_slices()` (in module `madminer.plotting.morphing`), 120
- `plot_pvalue_limits()` (in module `madminer.plotting.limits`), 118
- `plot_systematics()` (in module `madminer.plotting.uncertainties`), 120
- `plot_uncertainty()` (in module `madminer.plotting.uncertainties`), 121
- `profile_information()` (in module `madminer.fisherinformation.manipulate`), 53
- `profile_log_likelihood()` (in module `madminer.likelihood.manipulate`), 67
- `project_information()` (in module `madminer.fisherinformation.manipulate`), 54
- `project_log_likelihood()` (in module `madminer.likelihood.manipulate`), 68
- ## Q
- `QuadraticMorphingAwareRatioEstimator` (class in `madminer.ml.morphing_aware`), 101
- ## R
- `random_morphing_points()` (in module `madminer.sampling.parameters`), 125
- `rate_information()` (in module `madminer.fisherinformation.information.FisherInformation` method), 52
- `reset_cuts()` (`madminer.delphes.delphes_reader.DelphesReader` method), 33
- `reset_cuts()` (`madminer.lhe.lhe_reader.LHEReader` method), 59
- `reset_efficiencies()` (`madminer.lhe.lhe_reader.LHEReader` method), 59
- `reset_observables()` (`madminer.delphes.delphes_reader.DelphesReader` method), 33
- `reset_observables()` (`madminer.lhe.lhe_reader.LHEReader` method), 59
- `reweight_existing_sample()` (`madminer.core.madminer.MadMiner` method), 22
- `run()` (`madminer.core.madminer.MadMiner` method), 23
- `run_delphes()` (`madminer.delphes.delphes_reader.DelphesReader` method), 33
- `run_multiple()` (`madminer.core.madminer.MadMiner` method), 24
- ## S
- `sample_test()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 127
- `sample_train_density()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 129
- `sample_train_local()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 130
- `sample_train_more_ratios()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 131
- `sample_train_plain()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 133
- `sample_train_ratio()` (`madminer.sampling.sampleaugmenter.SampleAugmenter` method), 135
- `SampleAugmenter` (class in `madminer.sampling.sampleaugmenter`), 125
- `save()` (`madminer.core.madminer.MadMiner` method), 26
- `save()` (`madminer.delphes.delphes_reader.DelphesReader` method), 34
- `save()` (`madminer.lhe.lhe_reader.LHEReader` method), 59
- `save()` (`madminer.ml.base.ConditionalEstimator` method), 82
- `save()` (`madminer.ml.base.Estimator` method), 84
- `save()` (`madminer.ml.ensemble.Ensemble` method), 93
- `save()` (`madminer.ml.score.ScoreEstimator` method), 110
- `ScoreEstimator` (class in `madminer.ml.score`), 109
- `set_acceptance()` (`madminer.delphes.delphes_reader.DelphesReader` method), 34
- `set_benchmarks()` (`madminer.core.madminer.MadMiner` method), 26



`set_met_noise()` (*madminer.lhe.lhe\_reader.LHEReader* method), 60

`set_morphing()` (*madminer.core.madminer.MadMiner* method), 26

`set_nuisance()` (*madminer.ml.score.ScoreEstimator* method), 110

`set_parameters()` (*madminer.core.madminer.MadMiner* method), 27

`set_smearing()` (*madminer.lhe.lhe\_reader.LHEReader* method), 60

## T

`TheresAGoodReasonThisDoesntWork`, 84

`train()` (*madminer.ml.base.Estimator* method), 84

`train()` (*madminer.ml.double\_parameterized\_ratio.DoubleParameterizedRatioEstimator* method), 86

`train()` (*madminer.ml.likelihood.LikelihoodEstimator* method), 97

`train()` (*madminer.ml.morphing\_aware.MorphingAwareRatioEstimator* method), 99

`train()` (*madminer.ml.morphing\_aware.QuadraticMorphingAwareRatioEstimator* method), 102

`train()` (*madminer.ml.parameterized\_ratio.ParameterizedRatioEstimator* method), 106

`train()` (*madminer.ml.score.ScoreEstimator* method), 111

`train_all()` (*madminer.ml.ensemble.Ensemble* method), 93

`train_one()` (*madminer.ml.ensemble.Ensemble* method), 93

`truth_information()` (*madminer.fisherinformation.information.FisherInformation* method), 52

## W

`weighted_events()` (*madminer.analysis.dataanalyzer.DataAnalyzer* method), 16

## X

`xsec_gradients()` (*madminer.analysis.dataanalyzer.DataAnalyzer* method), 17

`xsecs()` (*madminer.analysis.dataanalyzer.DataAnalyzer* method), 17