ClusterKinG

Release 0.9.dev4

Jason Aebischer, Alejandro Celis, Kilian Lieret

Apr 17, 2019

First steps

1 Readme **3**
- 1.1 Description . . . 3
- 1.2 Physics Case . . . 3
- 1.3 Installation . . . 3
- 1.4 Usage and Documentation . . . 4
- 1.5 Example . . . 4
- 1.6 License & Contributing . . . 8

2 Data **9**
- 2.1 DFMD . . . 9
- 2.2 Data . . . 11
- 2.3 DataWithErrors . . . 16

3 Scanner **19**
- 3.1 Scanner . . . 19
- 3.2 WilsonScanner . . . 20

4 Cluster **23**
- 4.1 Cluster . . . 23
- 4.2 HierarchyCluster . . . 23
- 4.3 KmeansCluster . . . 24

5 Benchmark **25**
- 5.1 AbstractBenchmark . . . 25
- 5.2 Benchmark . . . 26

6 Plots **29**
- 6.1 ClusterPlot . . . 29
- 6.2 BundlePlot . . . 30

7 Maths **33**
- 7.1 Binning . . . 33
- 7.2 Metric . . . 33
- 7.3 Statistics . . . 34

8 Utility **37**
- 8.1 Interface . . . 37

8.2 Log 37
8.3 Metadata 38
8.4 Testing 39

9 Indices and tables **41**

Python Module Index **43**

The ClusterKinG package provides a flexible yet easy to use framework to cluster sets of histograms (or similar datasets) and to select benchmark points representing each cluster. The package particularly focuses on use cases in high energy physics.

CHAPTER 1

Readme

1.1 Description

This package provides a flexible yet easy to use framework to cluster sets of histograms (or other higher dimensional data) and to select benchmark points representing each cluster. The package particularly focuses on use cases in high energy physics.

1.2 Physics Case

While most of this package is very general and can be applied to a broad variety of use cases, we have been focusing on applications in high energy physics (particle physics) so far and provide additional convenience methods for this use case. In particular, most of the current tutorials are in this context.

Though very successful, the Standard Model of Particle Physics is believed to be uncomplete, prompting the search for New Physics (NP). The phenomenology of NP models typically depends on a number of free parameters, sometimes strongly influencing the shape of distributions of kinematic variables. Besides being an obvious challenge when presenting exclusion limits on such models, this also is an issue for experimental analyses that need to make assumptions on kinematic distributions in order to extract features of interest, but still want to publish their results in a very general way.

By clustering the NP parameter space based on a metric that quantifies the similarity of the resulting kinematic distributions, a small number of NP benchmark points can be chosen in such a way that they can together represent the whole parameter space. Experiments (and theorists) can then report exclusion limits and measurements for these benchmark points without sacrificing generality.

1.3 Installation

`clusterking` can be installed with the python package installer:

```
pip3 install clusterking
```

For a local installation, you might want to use the `--user` switch of `pip`. You can also update your current installation with `pip3 install --upgrade clusterking`.

For the latest development version type:

```
git clone https://github.com/clusterking/clusterking/
cd clusterking
pip3 install --user .
```

1.4 Usage and Documentation

Good starting point: **Jupyter notebooks** in the `examples/jupyter_notebook` directory (**run online using binder**).

For a documentation of the classes and functions in this package, **read the docs on readthedocs.io**.

1.5 Example

1.5.1 Sample and cluster

Being a condensed version of the basic tutorial, the following code is all that is needed to cluster the shape of the `q^2` distribution of `B-> D* tau nu` in the space of Wilson coefficients:

```
import flavio
import numpy as np
import clusterking as ck

s = ck.scan.WilsonScanner()
d = ck.DataWithErrors()

# Set up kinematic function

def dBrdq2(w, q):
  return flavio.sm_prediction("dBR/dq2(B+->Dtaunu)", q) + \
      flavio.np_prediction("dBR/dq2(B+->Dtaunu)", w, q)

s.set_dfunction(
  dBrdq2,
  binning=np.linspace(3.2, 11.6, 10),
  normalize=True
)

# Set sampling points in Wilson space

s.set_spoints_equidist(
  {
      "CVL_bctaunutau": (-1, 1, 10),
      "CSL_bctaunutau": (-1, 1, 10),
      "CT_bctaunutau": (-1, 1, 10)
  },
```

(continues on next page)

(continued from previous page)

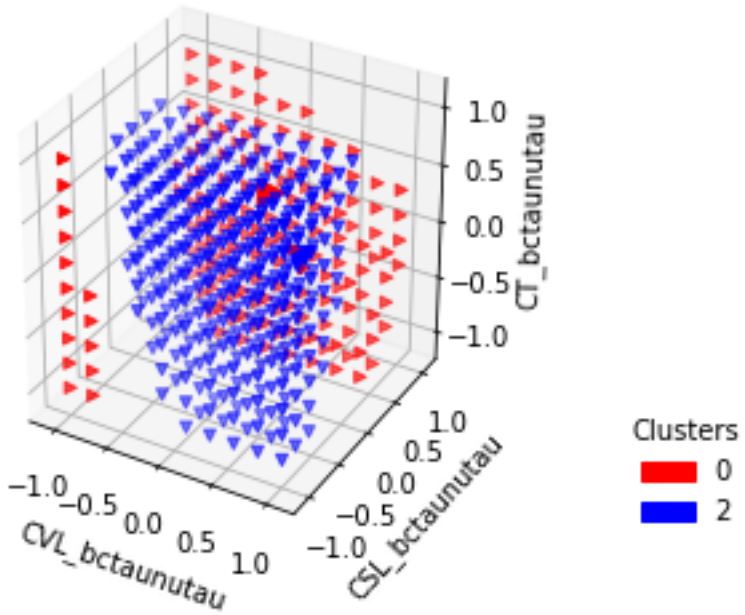
```
  scale=5,
  eft='WET',
  basis='flavio'
)

s.run(d)

# Use hierarchical clustering

c = ck.cluster.HierarchyCluster(d)
c.set_metric()          # Use default metric (Euclidean)
c.build_hierarchy()     # Build up clustering hierarchy
c.cluster(max_d=0.04)   # "Cut off" hierarchy
c.write()               # Write results to d
```

1.5.2 Benchmark points

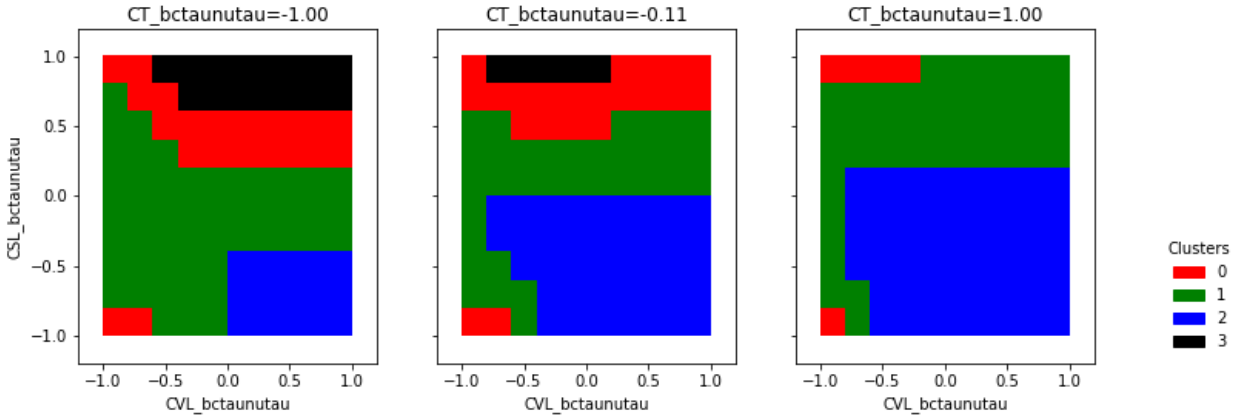
```
b = ck.Benchmark(d)
b.set_metric()       # Use default metric (Euclidean)
b.select_bpoints()   # Select benchmark points based on metric
b.write()            # Write results to d
```

1.5.3 Plotting

```
cp = ck.plots.ClusterPlot(d)
cp.scatter(
    ['CVL_bctaunutau', 'CSL_bctaunutau', 'CT_bctaunutau'],
    clusters=[1,2]  # Only plot 2 clusters for better visibility
)
```

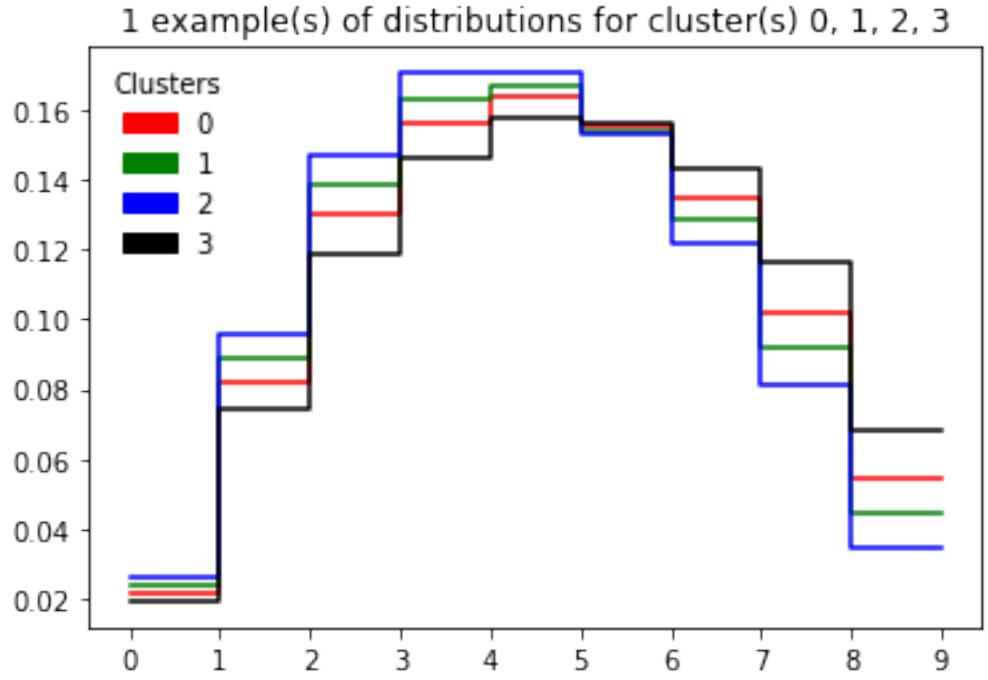


```
cp.fill(['CVL_bctaunutau', 'CSL_bctaunutau'])
```



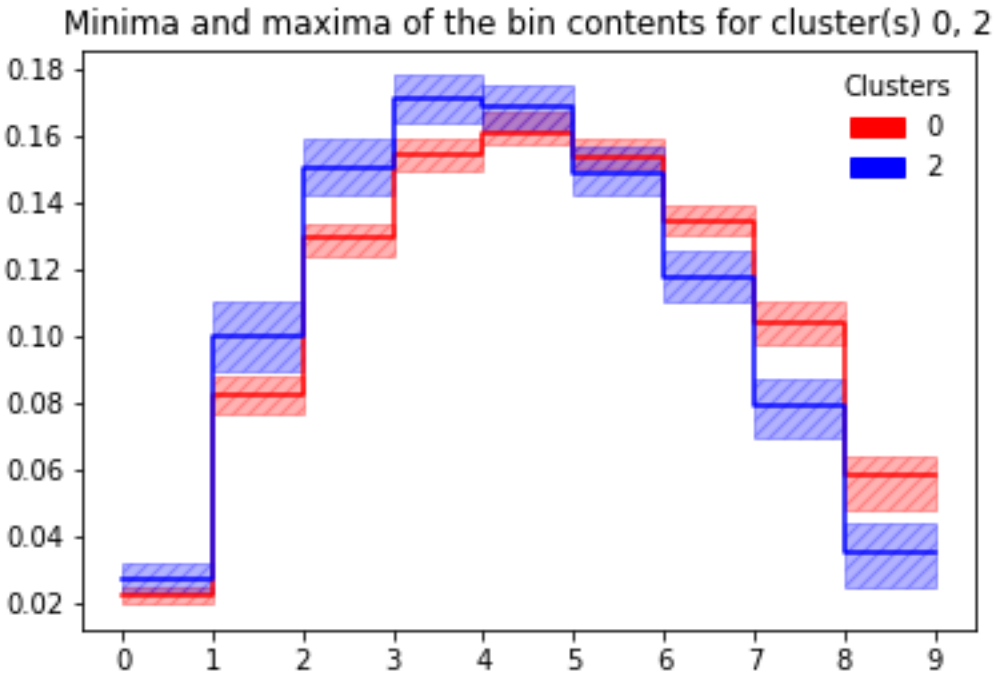
Plotting all benchmark points:

```
bp = ck.plots.BundlePlot(d)
bp.plot_bundles()
```



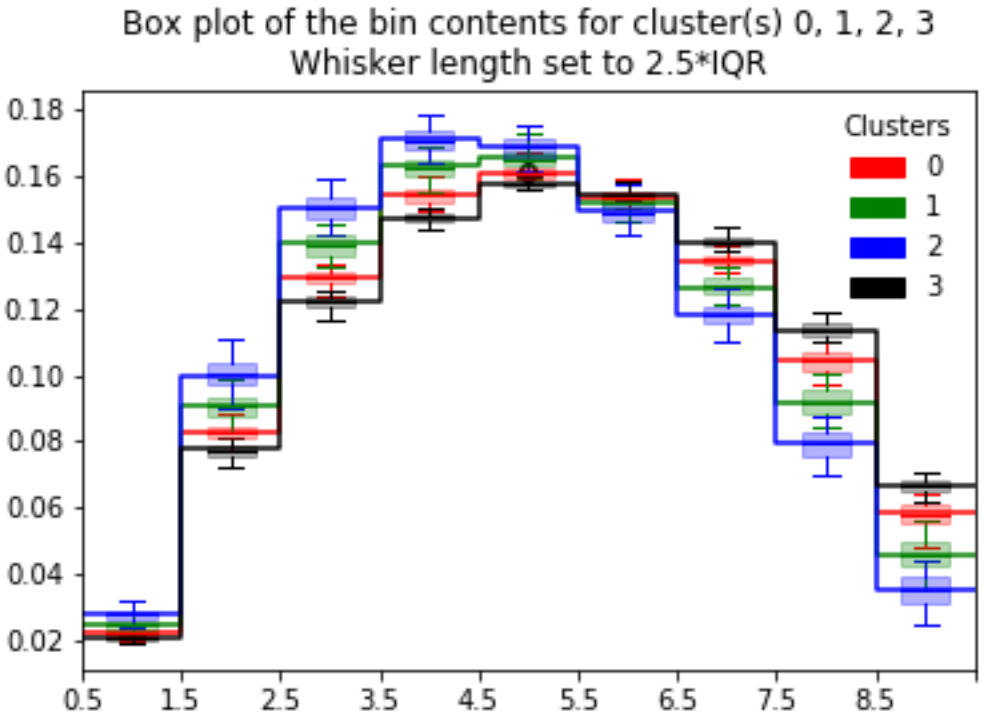
Plotting minima and maxima of bin contents for all histograms in a cluster (+benchmark histogram):

```
bp.plot_minmax(clusters=[0, 2])
```



Similarly with box plots:

```
bp.box_plot()
```



1.6 License & Contributing

This project is ongoing work and questions, comments, bug reports or pull requests are most welcome. You can also use the chat room on gitter or contact us via email. We are also working on a paper, so please make sure to cite us once we publish.

CHAPTER 2

Data

This page describes the main data object that are used by ClusterKinG. If you do not need to include errors in your analysis, use *Data*, else *DataWithErrors* (which inherits from *Data* but adds additional methods to it).

Both classes inherit from a very basic class, *DFMD*, which provides basic input and output methods.

2.1 DFMD

class clusterking.data.dfmd.**DFMD**(**args*, *log=None*, ***kwargs*)

Bases: object

This class bundles a pandas dataframe together with metadata and provides methods to load from and write these two to files.

__init__(**args*, *log=None*, ***kwargs*)

There are five different ways to initialize this class:

1. Initialize it empty: `DFMD()`.
2. From another DFMD object `my_dfmd`: `DFMD(my_dfmd)` or `DFMD(dfmd=my_dfmd)`.
3. From a directory path and a project name: `DFMD("path/to/io", "my_name")` or `DFMD(directory="path/to/io", name="my_name"`
4. From a dataframe and a metadata object (a nested dictionary like object) or paths to corresponding files: `DFMD(df=my_df, md=my_metadata)` or `DFMD(df="/path/to/df.csv", md=my_metadata)` etc.

> **Warning:** If you use `df=<pd.DataFrame>` or `md=<dict like>`, please be aware that this will not copy these objects, i.e. any changes that are done to these objects subsequently will affect both the original DataFrame/metadata and self.df or self.md. To avoid this, use `pd.DataFrame.copy()` or `dict.copy()` to create a deepcopy.

Parameters

- **log** – instance of `logging.Logger` or name of logger to be created
- ***args** – See above
- ****kwargs** – See above

log = None
instance of `logging.Logger`

df = None
Pandas dataframe to hold all of the results

md = None
This will hold all the configuration that we will write out

static get_df_path(*directory: Union[pathlib.PurePath, str], name: str*) → pathlib.Path
Return path to metadata json file based on directory and project name.

Parameters

- **directory** – Path to input/output directory
- **name** – Name of project

Returns Path to metadata json file.

static get_md_path(*directory: Union[pathlib.PurePath, str], name: str*) → pathlib.Path
Return path to dataframe csv file based on directory and project name.

Parameters

- **directory** – Path to input/output directory
- **name** – Name of project

Returns Path to dataframe csv file.

load_md(*md_path: Union[pathlib.PurePath, str]*) → None
Load metadata from json file generated by *write_md()*.

load_df(*df_path: Union[pathlib.PurePath, str]*) → None
Load dataframe from csv file creating by *write_md()*.

load(*directory: Union[pathlib.PurePath, str], name: str*) → None
Load from input files which have been generated from *write()*.

Parameters

- **directory** – Path to input/output directory
- **name** – Name of project

Returns None

write_md(*md_path: Union[pathlib.PurePath, str], overwrite='ask'*)
Write out metadata. The file can later be read in using *load_md()*.

Parameters

- **md_path** –
- **overwrite** – How to proceed if output file already exists: 'ask', 'overwrite', 'raise'

Returns:

write_df(*df_path*, *overwrite='ask'*)
Write out dataframe. The file can later be read in using *load_df()*.

Parameters

- **df_path** –
- **overwrite** – How to proceed if output file already exists: ‘ask’, ‘overwrite’, ‘raise’

Returns:

write(*directory: Union[pathlib.PurePath, str], name: str, overwrite='ask'*) → None
Write to input files that can be later loaded with *load()*.

Parameters

- **directory** – Path to input/output directory
- **name** – Name of project
- **overwrite** – How to proceed if output file already exists: ‘ask’, ‘overwrite’, ‘raise’

Returns:

copy(*deep=True*)
Make a copy of this object.

Parameters **deep** – Make a deep copy (default True). If this is disabled, any change to the copy will also affect the original.

Returns New object.

2.2 Data

class clusterking.data.data.**Data**(**args*, ***kwargs*)
Bases: *clusterking.data.dfmd.DFMD*

This class inherits from the *DFMD* class and adds additional methods to it. It is the basic container, that contains the

- The distributions to cluster
- The cluster numbers after clustering
- The benchmark points after they are selected.

__init__(**args*, ***kwargs*)

bin_cols
All columns that correspond to the bins of the distribution. This is automatically read from the metadata as set in e.g. the Scan.run.

par_cols
All columns that correspond to the parameters (e.g. Wilson parameters). This is automatically read from the metadata as set in e.g. the `clusterking.scan.scanner.Scanner.run()`.

n
Number of points in parameter space that were sampled.

nbins
Number of bins of the distribution.

npars
Number of parameters that were sampled (i.e. number of dimensions of the sampled parameter space.

data(*normalize=False*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dca26a0>
Returns all histograms as a large matrix.

Parameters **normalize** – Normalize all histograms

Returns numpy.ndarray of shape self.n x self.nbins

norms()
Returns a vector of all normalizations of all histograms (where each histogram corresponds to one sampled point in parameter space).

Returns numpy.ndarray of shape self.n

clusters(*cluster_column='cluster'*)
Return list of all cluster names (unique)

Parameters **cluster_column** – Column that contains the cluster names

get_param_values(*param: Union[None, str] = None*)
Return all unique values of this parameter

Parameters **param** – Name of parameter. If none is given, instead return a dictionary mapping of parameters to their values.

Returns:

only_bpoints(*bpoint_column='bpoint'*, *inplace=False*)
Keep only the benchmark points as sample points.

Parameters

- **bpoint_column** – benchmark point column (boolean)
- **inplace** – If True, the current Data object is modified, if False, a new copy of the Data object is returned.

Returns None or Data

fix_param(*inplace=False*, *bpoints=False*, *bpoint_slices=False*, *bpoint_column='bpoint'*, ***kwargs*)
Fix some parameter values to get a subset of sample points.

Parameters

- **inplace** – Modify this Data object instead of returning a new one
- **bpoints** – Keep bpoints (no matter if they are selected by the other selection or not)
- **bpoint_slices** – Keep all parameter values that are attained by benchmark points.
- **bpoint_column** – Column with benchmark points (default 'bpoints') (for use with the `bpoints` option)
- ****kwargs** – Specify parameter values: Use `<parameter name>=<value>` or `<parameter name>=[<value1>, ..., <valuen>]`.

Returns If `inplace == True`, return new Data with subset of sample points.

Examples:

```
d = Data("/path/to/tutorial/csv/folder", "tutorial_basics")
```

Return a new Data object, keeping the two values `CT_bctaunutau` closest to -0.75 or 0.5

```
d.fix_param(CT_bctaunutau=[-.75, 0.5])
```

Return a new Data object, where we also fix `CSL_bctaunutau` to the value closest to -1.0:

```
d.fix_param(CT_bctaunutau=[-.75, 0.5], CSL_bctaunutau=-1.0)
```

Return a new Data object, keeping the two values `CT_bctaunutau` closest to -0.75 or 0.5, but make sure we do not discard any benchmark points in that process:

```
d.fix_param(CT_bctaunutau=[-.75, 0.5], bpoints=True)
```

Return a new Data object, keeping the two values `CT_bctaunutau` closest to -0.75 or 0.5, but keep all values of `CT_bctaunutau` that are attained by at least one benchmark point:

```
d.fix_param(CT_bctaunutau=[-.75, 0.5], bpoint_slices=True)
```

Return a new Data object, keeping only those values of `CT_bctaunutau`, that are attained by at least one benchmark point:

```
d.fix_param(CT_bctaunutau=[], bpoint_slice=True)
```

sample_param(*bpoints=False*, *bpoint_slices=False*, *bpoint_column='bpoint'*, *inplace=False*, ***kwargs*)

Return a Data object that contains a subset of the sample points (points in parameter space). Similar to Data.fix_param.

Parameters

- **inplace** – Modify this Data object instead of returning a new one
- **bpoints** – Keep bpoints (no matter if they are selected by the other selection or not)
- **bpoint_slices** – Keep all parameter values that are attained by benchmark points
- **bpoint_column** – Column with benchmark points (default 'bpoints') (for use with the `bpoints` option)
- ****kwargs** – Specify parameter ranges: `<coeff name>=(min, max, npoints)` or `<coeff name>=npoints` For each coeff (identified by <coeff name>), select (at most) npoints points between min and max. In total this will therefore result in npoints_{coeff_1} x ... x npoints_{coeff_npar} sample points (provided that there are enough sample points available). If a coefficient isn't contained in the dictionary, this dimension of the sample remains untouched.

Returns If `inplace == True`, return new Data with subset of sample points.

Examples:

```
d = Data("/path/to/tutorial/csv/folder", "tutorial_basics")
```

Return a new Data object, keeping subsampling `CT_bctaunutau` closest to 5 values between -1 and 1:

```
d.sample_param(CT_bctaunutau=(-1, 1, 10))
```

The same in shorter syntax (because -1 and 1 are the minimum and maximum of the parameter)

```
d.sample_param(CT_bctaunutau=10)
```

For the `bpoints` and `bpoint_slices` syntax, see the documenation of *clusterking.data.data.Data.fix_param()*.

rename_clusters(*arg=None*, *column='cluster'*, *new_column=None*)

Rename clusters based on either

1. A dictionary of the form `{<old cluster name>: <new cluster name>}`
2. A function that maps the old cluster name to the new cluster name

Example for 2: Say our `Data` object `d` contains clusters 1 to 10 in the default column `cluster`. The following method call will instead use the numbers 0 to 9:

```
d.rename_clusters(lambda x: x-1)
```

Parameters

- **arg** – Dictionary or function as described above.
- **column** – Column that contains the cluster names
- **new_column** – New column to write to (default None, i.e. rename in place)

Returns None

plot_dist(*cluster_column='cluster'*, *bpoint_column='bpoint'*, *title=None*, *clusters=None*, *nlines=0*, *bpoints=True*, *legend=True*)

Plot several examples of distributions for each cluster specified.

Parameters

- **cluster_column** – Column with the cluster names (default 'cluster')
- **bpoint_column** – Column with bpoints (default 'bpoint')
- **title** – Plot title (`None`: automatic)
- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.
- **nlines** – Number of example distributions of each cluster to be plotted (default 0)
- **bpoints** – Draw benchmark points (default True)
- **legend** – Draw legend? (default True)

Note: To customize these kind of plots further, check the *BundlePlot* class and the *plot_bundles()* method thereof.

Returns Figure

plot_dist_minmax(*cluster_column='cluster'*, *bpoint_column='bpoint'*, *title=None*, *clusters=None*, *bpoints=True*, *legend=True*)

Plot the minimum and maximum of each bin for the specified clusters.

Parameters

- **cluster_column** – Column with the cluster names (default 'cluster')
- **bpoint_column** – Column with bpoints (default 'bpoint')
- **title** – Plot title (`None`: automatic)
- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.
- **bpoints** – Draw benchmark points (default True)
- **legend** – Draw legend? (default True)

Note: To customize these kind of plots further, check the *BundlePlot* class and the *plot_minmax()* method thereof.

Returns Figure

plot_dist_box(*cluster_column='cluster'*, *bpoint_column='bpoint'*, *title=None*, *clusters=None*, *bpoints=True*, *whiskers=2.5*, *legend=True*)

Box plot of the bin contents of the distributions corresponding to selected clusters.

Parameters

- **cluster_column** – Column with the cluster names (default 'cluster')
- **bpoint_column** – Column with bpoints (default 'bpoint')
- **title** – Plot title (`None`: automatic)
- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.
- **bpoints** – Draw benchmark points (default True)
- **whiskers** – Length of the whiskers of the box plot in units of IQR (interquartile range, containing 50% of all values). Default 2.5.
- **legend** – Draw legend? (default True)

Note: To customize these kind of plots further, check the *BundlePlot* class and the *box_plot()* method thereof.

Returns Figure

plot_clusters_scatter(*params*, *clusters=None*, *cluster_column='cluster'*, *bpoint_column='bpoint'*, *legend=True*, *max_subplots=16*, *max_cols=4*, *figsize=(4, 4)*, *markers=('o', 'v', '^', 'v', '<', '>')*)

Create scatter plot, specifying the columns to be on the axes of the plot. If 3 column are specified, 3D scatter plots are presented, else 2D plots. If the dataframe contains more columns, such that each row is not only specified by the columns on the axes, a selection of subplots is created, showing 'cuts'. Benchmark points are marked by enlarged plot markers.

Parameters

- **params** – The names of the columns to be shown on the x, y (and z) axis of the plots.
- **clusters** – The get_clusters to be plotted (default: all)
- **cluster_column** – Column with the cluster names (default 'cluster')
- **bpoint_column** – Column with bpoints (default 'bpoint')
- **legend** – Draw legend? (default True)
- **max_subplots** – Maximal number of subplots
- **max_cols** – Maximal number of columns of the subplot grid
- **figsize** – Figure size of each subplot
- **markers** – List of markers of the get_clusters

Returns Figure

plot_clusters_fill(*params*, *cluster_column='cluster'*, *bpoint_column='bpoint'*, *legend=True*, *max_subplots=16*, *max_cols=4*, *figsize=(4, 4)*)

Call this method with two column names, x and y. The results are similar to those of 2D scatter plots as created by the scatter method, except that the coloring is expanded to the whole xy plane. Note: This method only works with uniformly sampled NP!

Parameters

- **params** – The names of the columns to be shown on the x, y (and z) axis of the plots.

- **cluster_column** – Column with the cluster names (default 'cluster')
- **bpoint_column** – Column with bpoints (default 'bpoint')
- **legend** – Draw legend? (default True)
- **max_subplots** – Maximal number of subplots
- **max_cols** – Maximal number of columns of the subplot grid
- **figsize** – Figure size of each subplot

Returns:

2.3 DataWithErrors

class clusterking.data.dwe.**DataWithErrors**(**args*, ***kwargs*)

Bases: *clusterking.data.data.Data*

This class extends the `Data` class by convenient and performant ways to add errors to the distributions.

See the description of the `Data` class for more information about the data structure itself.

There are three basic ways to add errors: 1. Add relative errors (with correlation) relative to the bin content of each bin in the distribution: `add_rel_err_...` 2. Add absolute errors (with correlation): `add_err_...` 3. Add poisson errors: `add_err_poisson`

Note: All of these methods, add the errors in a consistent way for all sample points/distributions, i.e. it is impossible to add a certain error specifically to one sample point only!

Afterwards, you can get errors, correlation and covariance matrices for every data point by using one of the methods such as `cov`, `corr`, `err`.

Note: When saving your dataset, your error configuration is saved as well, so you can reload it like any other `Data` object.

Parameters **data** – n x nbins matrix

__init__(**args*, ***kwargs*)

rel_cov

abs_cov

poisson_errors

data(*decorrelate=False*, ***kwargs*)

Return data matrix

Parameters

- **decorrelate** – Unrotate the correlation matrix to return uncorrelated data entries
- ****kwargs** – Any keyword argument to `Data.data()`

Returns self.n x self.nbins array

cov(*relative=False*)

Return covariance matrix

Parameters **relative** – "Relative to data", i.e. $\mathrm{Cov}_{ij}/(\mathrm{data}_i \cdot \mathrm{data}_j)$

Returns self.n x self.nbins x self.nbins array

corr()
Return correlation matrix

Returns self.n x self.nbins x self.nbins array

err(*relative=False*)
Return errors per bin

Parameters **relative** – Relative errors

Returns self.n x self.nbins array

add_err_cov(*cov*) → None
Add error from covariance matrix.

Parameters **cov** – self.n x self.nbins x self.nbins array of covariance matrices or self.nbins x self.nbins covariance matrix (if equal for all data points)

add_err_corr(*err*, *corr*) → None
Add error from errors vector and correlation matrix.

Parameters

- **err** – self.n x self.nbins vector of errors for each data point and bin or self.nbins vector of uniform errors per data point or float (uniform error per bin and datapoint)
- **corr** – self.n x self.nbins x self.nbins correlation matrices or self.nbins x self.nbins correlation matrix

add_err_uncorr(*err*) → None
Add uncorrelated error.

Parameters **err** – see argument of *add_err_corr()*

add_err_maxcorr(*err*) → None
Add maximally correlated error.

Parameters **err** – see argument of *add_err_corr()*

add_rel_err_cov(*cov: <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dc9aa90>*) → None
Add error from "relative" covariance matrix

Parameters **cov** – see argument of *add_err_cov()*

add_rel_err_corr(*err*, *corr*) → None
Add error from relative errors and correlation matrix.

Parameters

- **err** – see argument of *add_err_corr()*
- **corr** – see argument of *add_err_corr()*

add_rel_err_uncorr(*err: <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dca26d8>*) → None
Add uncorrelated relative error.

Parameters **err** – see argument of *add_err_corr()*

add_rel_err_maxcorr(*err: <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dca2780>*) → None
Add maximally correlated relative error.

Parameters **err** – see argument of *add_err_corr()*

add_err_poisson(*scale=1*)
Add poisson errors/statistical errors.

Parameters **scale** – Scale poisson errors by this float (for example by your data normalization if your data itself is not normalized).

Returns None

CHAPTER 3

Scanner

3.1 Scanner

class clusterking.scan.**Scanner**

Bases: object

__init__()

spoints

Points in parameter space that are sampled.

set_dfunction(*func: Callable*, *binning: collections.abc.Sized = None*, *normalize=False*, ***kwargs*)

Set the function that generates the distributions that are later clustered (e.g. a differential cross section).

Parameters

- **func** – A function that takes the point in parameter space as the first argument. It should either return a float (if the binning option is specified), or a np.array elsewise.
- **binning** – If this parameter is not set (None), we will use the function as is. If it is set to an array-like object, we will integrate the function over the bins specified by this parameter.
- **normalize** – If a binning is specified, normalize the resulting distribution
- ****kwargs** – All other keyword arguments are passed to the function.

Returns None

> **Warning:** The function `func` has to be a globally defined function, else you will probably run into the error `Can't pickle local object ...` that is issued by the python multiprocessing module.

run(*data: clusterking.data.data.Data*, *no_workers=None*) → None

Calculate all sample points in parallel and saves the result in self.df.

Parameters

- **data** – Data object.
- **no_workers** – Number of worker nodes/cores. Default: Total number of cores.

3.2 WilsonScanner

class clusterking.scan.**WilsonScanner**

Bases: clusterking.scan.scanner.Scanner

Scans the NP parameter space in a grid and also in the kinematic variable.

Usage example:

```
import flavio
import functools
import numpy as np
import clusterking as ck

# Initialize Scanner object
s = ck.scan.WilsonScanner()

# Sample 4 points for each of the 5 Wilson coefficients
s.set_spoints_equidist(
    {
        "CVL_bctaunutau": (-1, 1, 4),
        "CSL_bctaunutau": (-1, 1, 4),
        "CT_bctaunutau": (-1, 1, 4)
    },
    scale=5,
    eft='WET',
    basis='flavio'
)

# Set function and binning
s.set_dfunction(
    functools.partial(flavio.np_prediction, "dBR/dq2(B+->Dtaunu)"),
    binning=np.linspace(3.15, 11.66, 10),
    normalize=True
)

# Initialize a Data objects to write to
d = ck.Data()

# Start running with maximally 3 cores and write the results to Data
s.run(d)
```

__init__()

set_spoints_grid(*values*, *scale*, *eft*, *basis*) → None

Set a grid of points in wilson space.

Parameters

- **values** – A dictionary of the following form:

```
{
    <wilson coeff name>: [
        value1,
```

(continues on next page)

(continued from previous page)

```
        value2,
        ...
    ]
}
```

- **scale** – Wilson coeff input scale in GeV
- **eft** – Wilson coeff input eft
- **basis** – Wilson coeff input basis

set_spoints_equidist(*ranges*, *scale*, *eft*, *basis*) → None
Set a list of 'equidistant' points in wilson space.

Parameters

- **ranges** – A dictionary of the following form:

```
{
    <wilson coeff name>: (
        <Minimum of wilson coeff>,
        <Maximum of wilson coeff>,
        <Number of bins between min and max>,
    )
}
```

- **scale** – <Wilson coeff input scale in GeV>,
- **eft** – <Wilson coeff input eft>,
- **basis** – <Wilson coeff input basis>

Returns None

CHAPTER 4

Cluster

4.1 Cluster

class clusterking.cluster.cluster.**Cluster**(*data: clusterking.data.data.Data*)
Bases: object

Abstract baseclass of the Cluster classes. This class is subclassed to implement specific clustering algorithms and defines common functions.

__init__(*data: clusterking.data.data.Data*)

md = None
Metadata

cluster(***kwargs*)
Performs the clustering. This method is a wrapper around the _cluster implementation in the subclasses. See there for additional arguments.

write(*cluster_column='cluster'*)
Write results back in the *Data* object.

4.2 HierarchyCluster

class clusterking.cluster.**HierarchyCluster**(*data*)
Bases: *clusterking.cluster.cluster.Cluster*

__init__(*data*)

metric = None
Function that, applied to Data or DWE object returns the metric as a condensed distance matrix.

set_metric(**args*, ***kwargs*) → None
Select a metric in one of the following ways:

1. If no positional arguments are given, we choose the euclidean metric.

2. If the first positional argument is string, we pick one of the metrics

   that are defined in `scipy.spatical.distance.pdist` by that name (all additional arguments will be past to this function).

3. If the first positional argument is a function, we take this function (and add all additional arguments to it).

Examples:

- `...()`: Euclidean metric
- `...("euclidean")`: Also Euclidean metric
- `...(lambda data: scipy.spatial.distance.pdist(data.data(), 'euclidean')`: Also Euclidean metric
- `...("minkowski", p=2)`: Minkowsky distance with `p=2`.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html for more information.

Parameters

- ***args** –
- ****kwargs** –

Returns Function that takes Data object as only parameter and returns a reduced distance matrix.

build_hierarchy(*method='complete'*, *optimal_ordering=False*) → None

Build the hierarchy object.

Parameters

- **method** – See reference on scipy.cluster.hierarchy.linkage
- **optimal_ordering** – See reference on scipy.cluster.hierarchy.linkage

dendrogram(*output: Union[None, str, pathlib.Path] = None*, *ax=None*, *show=False*, ***kwargs*) → Optional[<sphinx.ext.autodoc.importer._MockObject object at 0x7f178e02cb00>]

Creates dendrogram

Parameters

- **output** – If supplied, we save the dendrogram there
- **ax** – An axes object if you want to add the dendrogram to an existing axes rather than creating a new one
- **show** – If true, the dendrogram is shown in a viewer.
- ****kwargs** – Additional keyword options to scipy.cluster.hierarchy.dendrogram

Returns The matplotlib.pyplot.Axes object

4.3 KmeansCluster

class clusterking.cluster.**KmeansCluster**(*data*)

Bases: *clusterking.cluster.cluster.Cluster*

__init__(*data*)

CHAPTER 5

Benchmark

5.1 AbstractBenchmark

class clusterking.benchmark.abstract_benchmark.**AbstractBenchmark**(*data: clusterking.data.data.Data*, *cluster_column='cluster'*)

Bases: object

Subclass this class to implement algorithms to choose benchmark points from all the points (in parameter space) that correspond to one cluster.

__init__(*data: clusterking.data.data.Data*, *cluster_column='cluster'*)

Parameters

- **data** – *Data* object
- **cluster_column** – Column name of the clusters

cluster_column
The column from which we read the cluster information. Defaults to 'cluster'.

select_bpoints() → None
Select one benchmark point for each cluster.

write(*bpoint_column='bpoint'*) → None
Write benchmark points to a column in the dataframe of the data object.

Parameters **bpoint_column** – Column to write to

Returns None

5.2 Benchmark

class clusterking.benchmark.benchmark.**Benchmark**(*data*, *cluster_column='cluster'*)

Bases: *clusterking.benchmark.abstract_benchmark.AbstractBenchmark*

Selecting benchmarks based on a figure of merit that is calculated with the metric. You have to use `set_metric()` to specify the metric (as for the *HierarchyCluster* class). The default case for the figure of merit ("sum") chooses the point as benchmark point that minimizes the sum of all distances to all other points in the same cluster (where "distance" of course is with respect to the metric).

__init__(*data*, *cluster_column='cluster'*)

Parameters

- **data** – *Data* object
- **cluster_column** – Column name of the clusters

set_metric(**args*, ***kwargs*) → None

Select a metric in one of the following ways:

1. If no positional arguments are given, we choose the euclidean metric.
2. If the first positional argument is string, we pick one of the metrics

   that are defined in `scipy.spatical.distance.pdist` by that name (all additional arguments will be past to this function).

3. If the first positional argument is a function, we take this function (and add all additional arguments to it).

Examples:

- `...()`: Euclidean metric
- `...("euclidean")`: Also Euclidean metric
- `...(lambda data: scipy.spatial.distance.pdist(data.data(), 'euclidean')`: Also Euclidean metric
- `...("minkowski", p=2)`: Minkowsky distance with `p=2`.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html for more information.

Parameters

- ***args** –
- ****kwargs** –

Returns Function that takes Data object as only parameter and returns a reduced distance matrix.

set_fom(*fct: Callable*, **args*, ***kwargs*) → None

Set a figure of merit. The default case for the figure of merit ("sum") chooses the point as benchmark point that minimizes the sum of all distances to all other points in the same cluster (where "distance" of course is with respect to the metric). In general we choose the point that minimizes `self.fom(<metric>)`, i.e. the default case corresponds to `self.fom = lambda x: np.sum(x, axis=1)`, which you could have also set by calling `self.set_com(np.sum, axis=1)`.

Parameters

- **fct** – Function that takes the metric as first argument

- ***args** – Positional arguments that are added to the positional arguments of `fct` after the metric
- ****kwargs** – Keyword arguments for the function

Returns None

CHAPTER 6

Plots

6.1 ClusterPlot

class clusterking.plots.plot_clusters.**ClusterPlot**(*data*)

Bases: object

Plot clusters in parameter space.

After initialization, use the 'scatter' or 'fill' method for plotting.

You can modify the attributes of this class to tweak some properties of the plots.

__init__(*data*)

Parameters **data** – *Data* object

log = None
logging.Logger object

data = None
Instance of pandas.DataFrame

color_scheme = None
Color scheme

markers = None
List of markers of the get_clusters (scatter plot only).

max_subplots = None
Maximal number of subplots

max_cols = None
Maximal number of columns of the subplot grid

kv_formatter = None
Formatting of key-value pairs in title of plots

figsize = None
figure size of each subplot

cluster_column = None
The name of the column that holds the cluster index

bpoint_column = None
The name of the column that holds the benchmark yes/no information

default_marker_size = None
Default marker size

bpoint_marker_size = None
Marker size of benchmark points

draw_legend = None
If true, a legend is drawn

fig
The figure.

scatter(*cols: List[str], clusters=None, **kwargs*)
Create scatter plot, specifying the columns to be on the axes of the plot. If 3 column are specified, 3D scatter plots are presented, else 2D plots. If the dataframe contains more columns, such that each row is not only specified by the columns on the axes, a selection of subplots is created, showing 'cuts'. Benchmark points are marked by enlarged plot markers.

Parameters

- **cols** – The names of the columns to be shown on the x, y (and z) axis of the plots.
- **clusters** – The get_clusters to be plotted (default: all)
- ****kwargs** – Kwargs for ax.scatter

Returns The figure (unless the 'inline' setting of matplotllib is detected).

fill(*cols: List[str], kwargs_imshow=None*)
Call this method with two column names, x and y. The results are similar to those of 2D scatter plots as created by the scatter method, except that the coloring is expanded to the whole xy plane. Note: This method only works with uniformly sampled NP!

Parameters

- **cols** – List of name of column to be plotted on x-axis and on y-axis
- **kwargs_imshow** – Additional keyword arguments to be passed to imshow

Returns The figure (unless the 'inline' setting of matplotllib is detected).

savefig(**args*, ***kwargs*)
Equivalent to `ClusterPlot.fig.savefig(*args, **kwargs)`: Saves figure to file, e.g. `ClusterPlot.savefig("test.pdf")`.

6.2 BundlePlot

class clusterking.plots.plot_bundles.**BundlePlot**(*data*)
Bases: object

Plotting class to plot distributions by cluster in order to analyse which distributions get assigned to which cluster.

__init__(*data*)

Parameters **data** – *Data* object

log = None
logging.Logger object

data = None
pandas dataframe

cluster_column = None
Name of the column holding the cluster number

color_scheme = None
Color scheme

draw_legend = None
Draw legend?

title = None
Override default titles with this title. If None, the default title is used.

ax = None
Instance of matplotlib.axes.Axes

fig
Instance of matplotlib.pyplot.figure

plot_bundles(*clusters: Union[int, Iterable[int]] = None*, *nlines=0*, *ax=None*, *bpoints=True*) → None
Plot several examples of distributions for each cluster specified

Parameters

- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.
- **nlines** – Number of example distributions of each cluster to be plotted
- **ax** – Instance of matplotlib.axes.Axes to be plotted on. If None (default), a new axes object and figure is initialized and saved as self.ax and self.fig.
- **bpoints** – Draw benchmark curve

Returns None

animate_bundle(*cluster*, *n*, *benchmark=True*)

plot_minmax(*clusters: Union[int, Iterable[int]] = None*, *ax=None*, *bpoints=True*) → None
Plot the minimum and maximum of each bin for the specified clusters.

Parameters

- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.
- **ax** – Instance of `matplotlib.axes.Axes` to plot on. If None, a new one is instantiated.
- **bpoints** – Plot reference

Returns None

box_plot(*clusters: Union[int, Iterable[int]] = None*, *ax=None*, *whiskers=2.5*, *bpoints=True*) → None
Box plot of the bin contents of the distributions corresponding to selected clusters.

Parameters

- **clusters** – List of clusters to selected or single cluster. If None (default), all clusters are chosen.

- **ax** – Instance of matplotlib.axes.Axes to plot on. If None, a new one is instantiated.
- **whiskers** – Length of the whiskers of the box plot in units of IQR (interquartile range, containing 50% of all values). Default 2.5.
- **bpoints** – Draw benchmarks?

CHAPTER 7

Maths

Mathematics.

7.1 Binning

clusterking.maths.binning.**bin_function**(*fct*, *binning: <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dc41240>*, *normalize=False*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f178dc41278>

Bin function, i.e. calculate the integrals of a function for each bin.

Parameters

- **fct** – Function to be integrated per bin
- **binning** – Array of bin edge points.
- **normalize** – If true, we will normalize the distribution, i.e. divide by the sum of all bins in the end.

Returns Array of bin contents

7.2 Metric

clusterking.maths.metric.**condense_distance_matrix**(*matrix*)

Convert a square-form distance matrix to a vector-form distance vector

Parameters **matrix** – n x n symmetric matrix with 0 diagonal

Returns n choose 2 vector

clusterking.maths.metric.**uncondense_distance_matrix**(*vector*)

Convert a vector-form distance vector to a square-form distance matrix

Parameters **vector** – n choose 2 vector

Returns n x n symmetric matrix with 0 diagonal

clusterking.maths.metric.**metric_selection**(**args*, ***kwargs*) → Callable

Select a metric in one of the following ways:

1. If no positional arguments are given, we choose the euclidean metric.
2. If the first positional argument is string, we pick one of the metrics

   that are defined in `scipy.spatical.distance.pdist` by that name (all additional arguments will be past to this function).

3. If the first positional argument is a function, we take this function (and add all additional arguments to it).

Examples:

- `...()`: Euclidean metric
- `...("euclidean")`: Also Euclidean metric
- `...(lambda data: scipy.spatial.distance.pdist(data.data(), 'euclidean')`: Also Euclidean metric
- `...("minkowski", p=2)`: Minkowsky distance with `p=2`.

See https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html for more information.

Parameters

- ***args** –
- ****kwargs** –

Returns Function that takes Data object as only parameter and returns a reduced distance matrix.

clusterking.maths.metric.**chi2_metric**(*dwe: clusterking.data.dwe.DataWithErrors*, *output='condensed'*)

Returns the chi2/ndf values of the comparison of a datasets.

Parameters

- **dwe** –
- **output** – 'condensed' (condensed distance matrix) or 'full' (full distance matrix)

Returns Condensed distance matrix

7.3 Statistics

clusterking.maths.statistics.**ensure_array**(*x*)

clusterking.maths.statistics.**cov2err**(*cov*)

Convert covariance matrix (or array of covariance matrices of equal shape) to error array (or array thereof).

Parameters **cov** – [n x] nbins x nbins array

Returns [n x] nbins array

clusterking.maths.statistics.**cov2corr**(*cov*)

Convert covariance matrix (or array of covariance matrices of equal shape) to correlation matrix (or array thereof).

Parameters **cov** – [n x] nbins x nbins array

Returns [n x] nbins x nbins array

clusterking.maths.statistics.**corr2cov**(*corr*, *err*)

Convert correlation matrix (or array of covariance matrices of equal shape) together with error array (or array thereof) to covariance matrix (or array thereof).

Parameters

- **corr** – [n x] nbins x nbins array
- **err** – [n x] nbins array

Returns [n x] nbins x nbins array

clusterking.maths.statistics.**rel2abs_cov**(*cov*, *data*)

Convert relative covariance matrix to absolute covariance matrix

Parameters

- **cov** – n x nbins x nbins array
- **data** – n x nbins array

Returns n x nbins x nbins array

clusterking.maths.statistics.**abs2rel_cov**(*cov*, *data*)

Convert covariance matrix to relative covariance matrix

Parameters

- **cov** – n x nbins x nbins array
- **data** – n x nbins array

Returns n x nbins x nbins array

CHAPTER 8

Utility

This module bundles mostly technical utilities that might not be all this interesting for users.

8.1 Interface

Utils for the command line interface (CLI).

clusterking.util.cli.**yn_prompt**(*question: str*, *yes=None*, *no=None*) → bool
Ask yes-no question.

Parameters

- **question** – Description of the prompt
- **yes** – List of strings interpreted as yes
- **no** – List of strings interpreted as no

Returns True if yes, False if no.

clusterking.util.cli.**handle_overwrite**(*paths*, *behavior*, *log*)
Do we want to overwrite a file that exists?

Parameters

- **paths** – List of pathlib.Paths
- **behavior** – How to proceed if output file already exists: 'ask', 'overwrite', 'raise'
- **log** – logging.Logger instance

Returns True if overwrite will occurr, False otherwise.

8.2 Log

Defines an easy function to set up a logger.

clusterking.util.log.**get_logger**(*name='Logger'*, *level=10*, *sh_level=10*)
Sets up a logging.Logger.

If the colorlog module is available, the logger will use colors, otherwise it will be in b/w. The colorlog module is available at https://github.com/borntyping/python-colorlog but can also easily be installed with e.g. 'sudo pip3 colorlog' or similar commands.

Parameters

- **name** – name of the logger
- **level** – General logging level
- **sh_level** – Logging level of stream handler

Returns Logger

clusterking.util.log.**silence_all_logs**(*level=30*)

8.3 Metadata

Miscellaneous utilities

clusterking.util.metadata.**nested_dict**()
This is very clever and stolen from https://stackoverflow.com/questions/16724788/ Use it to initialize a dictionary-like object which automatically adds levels. E.g.

```
a = nested_dict()
a['test']['this']['is']['working'] = "yaaay"
```

clusterking.util.metadata.**git_info**(*log=None*, *path=None*) → Dict[str, str]
Return dictionary containing status of the git repository (commit hash, date etc.

Parameters

- **log** – logging.Logger object (optional)
- **path** – path to .git subfolder or search path (optional)

Returns dictionary

clusterking.util.metadata.**save_git_info**(*output_path=None*, **args*, ***kwargs*) → Dict[str, str]
Save output of git_info to a file.

Parameters

- **output_path** – Output path. If None, the default will be bclustering/git_info.json
- ***args** – Passed on to git_info
- ****kwargs** – Passed on to git_info

Returns Output of git_info

clusterking.util.metadata.**load_git_info**(*input_path=None*) → Dict[str, str]
Load previously saved output of git_info from a json file.

Parameters **input_path** – Input path to json file. If None, the default will be bclustering/git_info.json

Returns Parsed json file (should be identical to saved output of git_info).

clusterking.util.metadata.**failsafe_serialize**(*obj*)

8.4 Testing

clusterking.util.testing.**set_testing_mode**(*testing_mode: bool*) → None
Set an environment variable signalling if we are in testing mode.

Parameters **testing_mode** (*bool*) – True if we are in testing mode

Returns None

clusterking.util.testing.**is_testing_mode**()

clusterking.util.testing.**test_jupyter_notebook**(*path*) → None
Runs jupyter notebook. A ValueError is raised if the file was not found.

class clusterking.util.testing.**MyTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Implements an additional general testing methods.

assertAllClose(*a*, *b*)
Compares two numpy arrays

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

clusterking.maths.binning, 33
clusterking.maths.metric, 33
clusterking.maths.statistics, 34
clusterking.util.cli, 37
clusterking.util.log, 37
clusterking.util.metadata, 38
clusterking.util.testing, 39

Index

Symbols

__init__() (*clusterking.benchmark.abstract_benchmark.AbstractBenchmark method*), 25
__init__() (*clusterking.benchmark.benchmark.Benchmark method*), 26
__init__() (*clusterking.cluster.HierarchyCluster method*), 23
__init__() (*clusterking.cluster.KmeansCluster method*), 24
__init__() (*clusterking.cluster.cluster.Cluster method*), 23
__init__() (*clusterking.data.data.Data method*), 11
__init__() (*clusterking.data.dfmd.DFMD method*), 9
__init__() (*clusterking.data.dwe.DataWithErrors method*), 16
__init__() (*clusterking.plots.plot_bundles.BundlePlot method*), 30
__init__() (*clusterking.plots.plot_clusters.ClusterPlot method*), 29
__init__() (*clusterking.scan.Scanner method*), 19
__init__() (*clusterking.scan.WilsonScanner method*), 20

A

abs2rel_cov() (*in module clusterking.maths.statistics*), 35
abs_cov (*clusterking.data.dwe.DataWithErrors attribute*), 16
AbstractBenchmark (*class in clusterking.benchmark.abstract_benchmark*), 25
add_err_corr() (*clusterking.data.dwe.DataWithErrors method*), 17
add_err_cov() (*clusterking.data.dwe.DataWithErrors method*), 17
add_err_maxcorr() (*clusterking.data.dwe.DataWithErrors method*), 17
add_err_poisson() (*clusterking.data.dwe.DataWithErrors method*), 18
add_err_uncorr() (*clusterking.data.dwe.DataWithErrors method*), 17
add_rel_err_corr() (*clusterking.data.dwe.DataWithErrors method*), 17
add_rel_err_cov() (*clusterking.data.dwe.DataWithErrors method*), 17
add_rel_err_maxcorr() (*clusterking.data.dwe.DataWithErrors method*), 17
add_rel_err_uncorr() (*clusterking.data.dwe.DataWithErrors method*), 17
animate_bundle() (*clusterking.plots.plot_bundles.BundlePlot method*), 31
assertAllClose() (*clusterking.util.testing.MyTestCase method*), 39
ax (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31

B

Benchmark (*class in clusterking.benchmark.benchmark*), 26
bin_cols (*clusterking.data.data.Data attribute*), 11
bin_function() (*in module clusterking.maths.binning*), 33
box_plot() (*clusterking.plots.plot_bundles.BundlePlot method*), 31
bpoint_column (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 30
bpoint_marker_size (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 30
build_hierarchy() (*clusterking.cluster.HierarchyCluster method*), 24
BundlePlot (*class in clusterking.plots.plot_bundles*), 30

C

chi2_metric() (*in module clusterking.maths.metric*), 34
Cluster (*class in clusterking.cluster.cluster*), 23
cluster() (*clusterking.cluster.cluster.Cluster method*), 23
cluster_column (*clusterking.benchmark.abstract_benchmark.AbstractBenchmark attribute*), 25
cluster_column (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31
cluster_column (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
clusterking.maths.binning (*module*), 33
clusterking.maths.metric (*module*), 33
clusterking.maths.statistics (*module*), 34
clusterking.util.cli (*module*), 37
clusterking.util.log (*module*), 37
clusterking.util.metadata (*module*), 38
clusterking.util.testing (*module*), 39
ClusterPlot (*class in clusterking.plots.plot_clusters*), 29
clusters() (*clusterking.data.data.Data method*), 12
color_scheme (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31
color_scheme (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
condense_distance_matrix() (*in module clusterking.maths.metric*), 33
copy() (*clusterking.data.dfmd.DFMD method*), 11
corr() (*clusterking.data.dwe.DataWithErrors method*), 17
corr2cov() (*in module clusterking.maths.statistics*), 35
cov() (*clusterking.data.dwe.DataWithErrors method*), 16
cov2corr() (*in module clusterking.maths.statistics*), 34
cov2err() (*in module clusterking.maths.statistics*), 34

D

Data (*class in clusterking.data.data*), 11
data (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31
data (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
data() (*clusterking.data.data.Data method*), 11
data() (*clusterking.data.dwe.DataWithErrors method*), 16
DataWithErrors (*class in clusterking.data.dwe*), 16
default_marker_size (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 30
dendrogram() (*clusterking.cluster.HierarchyCluster method*), 24
df (*clusterking.data.dfmd.DFMD attribute*), 10
DFMD (*class in clusterking.data.dfmd*), 9
draw_legend (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31
draw_legend (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 30

E

ensure_array() (*in module clusterking.maths.statistics*), 34
err() (*clusterking.data.dwe.DataWithErrors method*), 17

F

failsafe_serialize() (*in module clusterking.util.metadata*), 38
fig (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31
fig (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 30
figsize (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
fill() (*clusterking.plots.plot_clusters.ClusterPlot method*), 30
fix_param() (*clusterking.data.data.Data method*), 12

G

get_df_path() (*clusterking.data.dfmd.DFMD static method*), 10
get_logger() (*in module clusterking.util.log*), 37
get_md_path() (*clusterking.data.dfmd.DFMD static method*), 10
get_param_values() (*clusterking.data.data.Data method*), 12
git_info() (*in module clusterking.util.metadata*), 38

H

handle_overwrite() (*in module clusterking.util.cli*), 37
HierarchyCluster (*class in clusterking.cluster*), 23

I

is_testing_mode() (*in module clusterking.util.testing*), 39

K

KmeansCluster (*class in clusterking.cluster*), 24

kv_formatter (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29

L

load() (*clusterking.data.dfmd.DFMD method*), 10
load_df() (*clusterking.data.dfmd.DFMD method*), 10
load_git_info() (*in module clusterking.util.metadata*), 38
load_md() (*clusterking.data.dfmd.DFMD method*), 10
log (*clusterking.data.dfmd.DFMD attribute*), 10
log (*clusterking.plots.plot_bundles.BundlePlot attribute*), 30
log (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29

M

markers (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
max_cols (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
max_subplots (*clusterking.plots.plot_clusters.ClusterPlot attribute*), 29
md (*clusterking.cluster.cluster.Cluster attribute*), 23
md (*clusterking.data.dfmd.DFMD attribute*), 10
metric (*clusterking.cluster.HierarchyCluster attribute*), 23
metric_selection() (*in module clusterking.maths.metric*), 34
MyTestCase (*class in clusterking.util.testing*), 39

N

n (*clusterking.data.data.Data attribute*), 11
nbins (*clusterking.data.data.Data attribute*), 11
nested_dict() (*in module clusterking.util.metadata*), 38
norms() (*clusterking.data.data.Data method*), 12
npars (*clusterking.data.data.Data attribute*), 11

O

only_bpoints() (*clusterking.data.data.Data method*), 12

P

par_cols (*clusterking.data.data.Data attribute*), 11
plot_bundles() (*clusterking.plots.plot_bundles.BundlePlot method*), 31
plot_clusters_fill() (*clusterking.data.data.Data method*), 15
plot_clusters_scatter() (*clusterking.data.data.Data method*), 15
plot_dist() (*clusterking.data.data.Data method*), 14
plot_dist_box() (*clusterking.data.data.Data method*), 14
plot_dist_minmax() (*clusterking.data.data.Data method*), 14
plot_minmax() (*clusterking.plots.plot_bundles.BundlePlot method*), 31
poisson_errors (*clusterking.data.dwe.DataWithErrors attribute*), 16

R

rel2abs_cov() (*in module clusterking.maths.statistics*), 35
rel_cov (*clusterking.data.dwe.DataWithErrors attribute*), 16
rename_clusters() (*clusterking.data.data.Data method*), 13
run() (*clusterking.scan.Scanner method*), 19

S

sample_param() (*clusterking.data.data.Data method*), 13
save_git_info() (*in module clusterking.util.metadata*), 38
savefig() (*clusterking.plots.plot_clusters.ClusterPlot method*), 30
Scanner (*class in clusterking.scan*), 19
scatter() (*clusterking.plots.plot_clusters.ClusterPlot method*), 30
select_bpoints() (*clusterking.benchmark.abstract_benchmark.AbstractBenchmark method*), 25
set_dfunction() (*clusterking.scan.Scanner method*), 19
set_fom() (*clusterking.benchmark.benchmark.Benchmark method*), 26
set_metric() (*clusterking.benchmark.benchmark.Benchmark method*), 26
set_metric() (*clusterking.cluster.HierarchyCluster method*), 23
set_spoints_equidist() (*clusterking.scan.WilsonScanner method*), 21
set_spoints_grid() (*clusterking.scan.WilsonScanner method*), 20
set_testing_mode() (*in module clusterking.util.testing*), 39
silence_all_logs() (*in module clusterking.util.log*), 38
spoints (*clusterking.scan.Scanner attribute*), 19

T

test_jupyter_notebook() (*in module clusterking.util.testing*), 39
title (*clusterking.plots.plot_bundles.BundlePlot attribute*), 31

U

uncondense_distance_matrix() (*in module clusterking.maths.metric*), 33

W

WilsonScanner (*class in clusterking.scan*), 20
write() (*clusterking.benchmark.abstract_benchmark.AbstractBenchmark method*), 25
write() (*clusterking.cluster.cluster.Cluster method*), 23
write() (*clusterking.data.dfmd.DFMD method*), 11
write_df() (*clusterking.data.dfmd.DFMD method*), 10
write_md() (*clusterking.data.dfmd.DFMD method*), 10

Y

yn_prompt() (*in module clusterking.util.cli*), 37