HErmes Documentation

Release 0.7.0

Achim Stoessl

Sep 19, 2018

Contents

What is an event selection?

In the context of high energy physics, event selection means the enhancement of the signal-to-noise rate by implementing filter criteria on the data. Since the signal consists of individual "events" (like a collision of particles in a collider) selecting only events which appear to be "signal-like" as defined by certain criteria is one of the basic tasks for a typical analysis in high energy physics. Typically the number of these kinds of events is very small compared to the number of background events (which are not "interesting" to the respective analyzer).

How can this package help with the task?

Selecting events is easy. However what is more complicated is the bookkeeping. To illustrate this, we have to go a bit more into the details:

First, let's start with some definitions:

- A **variable** describes a quantity which can describe signalness, e.g. energy.
- A **cut** describes a quality criterion, which is a condition imposed on a variable, e.g. "All events with energies larger then 100TeV"
- A data **category** is given by the fact that in many cases there is more than one type of data of interest which have to be studied simultaniously. For example this can be:
  - Real data, and a simulation of the signal and background
  - Different types of signal and background simulations for different kinds of hypothesis
  - Different types of data, e.g. different years of experimental data which need to be compared.

  and so on...
- A **dataset** means in this context a compilation of categories.

With these definitions, it is now possible to talk about **bookkeeping**: it is simply the necessity to ensure that every cut which is done the same way on each category of a dataset. This software intends to perform this task as painless as possible.

Another problem: fragmented datasources..

Often times, the data does not reach the analyzer in a consistent way: There might be several data files for a category, or different names for a variable. This software fixes some of these issues.

Why not just use root?

Root is certainly the most popular framework used in particle physics. The here described package does not intend to reimplement all the statistical and physics oriented features of root. The HErmes toolset allows for a quick inspection of a dataset and pre-analysis with the focus of questions like: "How well does my simulation agree with data?" or "What signal rate can I expect from a certain dataset?". If questions like that need to be accessed quickly, then this package might be helpful. For elaborated analysis tools, other software (like Root) might be a better choice.

The *HErmes* package is especially optimized to make the step from a bunch of files to a distribution after applications of some cuts as painless as possible.

CHAPTER 1

HErmes documentation contents

1.1 HErmes package

1.1.1 Subpackages

HErmes.analysis package

Submodules

HErmes.analysis.calculus module

Common calculations

HErmes.analysis.calculus.**opening_angle**(*reco_zen*, *reco_azi*, *true_zen*, *true_azi*)

Calculate the opening angle between two vectors, described by azimuth and zenith in some coordinate system. Can be useful for estimatiion of angular uncertainty for some reconstruction. Zenith and Azimuth in radians.

Parameters

- **reco_zen** (*float*) – zenith of vector A
- **reco_azi** (*float*) – azimuth of vector A
- **true_zen** (*float*) – zenith of vector B
- **true_azi** (*float*) – azimuth of vector B

Returns Opening angle in degree

Return type float

HErmes.analysis.fluxes module

Models for particle fluxes. These are just examples, for specific cosmic ray modelss have a look at e.g. https://github.com/afedynitch/CRFluxModels.git

class HErmes.analysis.fluxes.**Constant**
Bases: object

static identity(*x*)

class HErmes.analysis.fluxes.**PowerLawFlux**(*emin*, *emax*, *phi0*, *gamma*)
Bases: object

A flux only dependent on the energy of a particle, following a power law. Defined in an energy interval [emin, emax] with fluence phi0 and spectral index gamma

static E2_1E8(*energy*)
A flux with fixed parameters, spectral index E**-2 and normalization 1E-8 Usefull for automatic weighting.

Parameters **energy** –

fluxsum()
The integrated flux

Returns float

HErmes.analysis.tasks module

Investigate variables

HErmes.analysis.tasks.**construct_slices**(*name*, *bins*)
Prepare a set of cuts for the variable with name"name" in the dataset

Parameters

- **name** (*str*) – The name of the variable in the dataset
- **bins** (*array*) – bincenters of the slices

Returns tuple (list of strings, list of cuttuples)

Module contents

A compilation of analysis relevated calculus and physics tools

HErmes.fitting package

Submodules

HErmes.fitting.fit module

Provide routines for fitting charge histograms

HErmes.fitting.fit.**fit_model**(*charges*, *model*, *startparams=None*, *rej_outliers=False*, *nbins=200*, *silent=False*, *parameter_text=(('$\\mu_{{SPE}}$& {:4.2e}\\\\', 5),)*, *use_minuit=False*, *normalize=True*, ***kwargs*)
Standardazied fitting routine

Parameters

- **charges** (*np.ndarray*) – Charges obtained in a measurement (no histogram)
- **model** (*pyosci.fit.Model*) – A model to fit to the data
- **startparams** (*tuple*) – initial parameters to model, or None for first guess

Keyword Arguments

- **rej_outliers** (*bool*) – Remove extreme outliers from data
- **nbins** (*int*) – Number of bins
- **parameter_text** (*tuple*) – will be passed to model.plot_result
- **use_miniuit** (*bool*) – use minuit to minimize startparams for best chi2
- **normalize** (*bool*) – normalize data before fitting
- **silent** (*bool*) – silence output

Returns tuple

HErmes.fitting.fit.**reject_outliers**(*data*, *m=2*)
A simple way to remove extreme outliers from data

Parameters

- **data** (*np.ndarray*) – data with outliers
- **m** (*int*) – number of standard deviations outside the data should be discarded

Returns np.ndarray

HErmes.fitting.functions module

Provide some simple functions which can be used to create models

HErmes.fitting.functions.**calculate_chi_square**(*data*, *model_data*)
Very simple estimator for goodness-of-fit. Use with care. Non normalized bin counts are required.

Parameters

- **data** (*np.ndarray*) – observed data (bincounts)
- **model_data** (*np.ndarray*) – model predictions for each bin

Returns np.ndarray

HErmes.fitting.functions.**calculate_sigma_from_amp**(*amp*)
Get the sigma for the gauss from its peak value. Gauss is normed

Parameters **amp** (*float*) –

Returns float

HErmes.fitting.functions.**exponential**(*x*, *lmbda*)
An exponential model, e.g. for a decay with coefficent lmbda.

Parameters

- **x** (*float*) – input
- **lmbda** (*float*) – The exponent of the exponential

Returns np.ndarray

HErmes.fitting.functions.**gauss**(*x*, *mu*, *sigma*)
Returns a normed gaussian.

Parameters

- **x** (*np.ndarray*) – x values
- **mu** (*float*) – Gauss mu
- **sigma** (*float*) – Gauss sigma
- **n** –

Returns:

HErmes.fitting.functions.**n_gauss**(*x*, *mu*, *sigma*, *n*)
Returns a normed gaussian in the case of n ==1. If n > 1, The gaussian mean is shifted by n and its width is enlarged by the factor of n. The envelope of a sequence of these gaussians will be an expoenential.

Parameters

- **x** (*np.ndarray*) – x values
- **mu** (*float*) – Gauss mu
- **sigma** (*float*) – Gauss sigma
- **n** (*int*) – > 0, linear coefficient

Returns:

HErmes.fitting.functions.**pandel_factory**(*c_ice*)
Create a pandel function with the defined parameters

Parameters **c_ice** (*float*) – group velocity in ice in m/ns

Returns callable

HErmes.fitting.functions.**poisson**(*x*, *lmbda*)
Poisson probability

Parameters

- **x** (*int*) – measured number of occurences
- **lmbda** (*int*) – expected number of occurences

Returns np.ndarray

HErmes.fitting.model module

Provide a simple, easy to use model for fitting data and especially distributions

class HErmes.fitting.model.**Model**(*func*, *startparams=None*, *limits=((-inf, inf),)*, *errors=(10.0,)*, *func_norm=1*)
Bases: object

Model data with a parametrized prediction

add_data(*data*, *bins=200*, *create_distribution=False*, *normalize=False*, *density=True*, *xs=None*, *subtract=None*)
Add some data to the model, in preparation for the fit

Parameters **data** (*np.array*) –

Keyword Args nbins (int): subtract (callable): normalize (bool): normalize the data before adding density (bool): if normalized, assume the data is a pdf.

if False, use bincount for normalization.

Returns:

add_first_guess(*func*)
Use func to estimate better startparameters

Parameters **func** – Has to yield a set of startparameters

Returns:

clear()
Reset the model

Returns None

components

construct_error_function(*startparams*, *errors*, *limits*, *errordef*)

couple_all_models()
Use the first models startparams for the combined model

Returns None

couple_models(*coupling_variable*)
Couple the models by a variable, which means use the variable not independently in all model components, but fit it only once. E.g. if there are 3 models with parameters p1, p2, k each and they are coupled by k, parameters p11, p21, p12, p22, and k will be fitted instead of p11, p12, k1, p21, p22, k2.

Parameters **coupling_variable** – variable number of the number in startparams

Returns None

distribution

eval_first_guess(*data*)
Assign a new set of start parameters obtained by calling the first geuss metthod

Parameters **data** –

Returns

extract_parameters()
Get the variable names and coupling references for the individual model components

Returns tuple

fit_to_data(*silent=False*, *use_minuit=True*, *errors=None*, *limits=None*, *errordef=1000*, ***kwargs*)
Apply this model to data

Parameters

- **data** (*np.ndarray*) – the data, unbinned
- **silent** (*bool*) – silence output
- **use_minuit** (*bool*) – use minuit for fitting
- **errors** (*list*) – errors for minuit, see miniuit manual
- **limits** (*list of tuples*) – limits for minuit, see minuit manual
- **errordef** (*int*) – convergence criterion, see minuit manual

- ****kwargs** – will be passed on to scipy.optimize.curvefit

Returns None

n_free_params
The number of free parameters of this model

Returns int

plot_result(*ymin=1000*, *xmax=8*, *ylabel='normed bincount'*, *xlabel='Q [C]'*, *fig=None*, *log=True*, *axes_range='auto'*, *model_alpha=0.3*, *add_parameter_text=(('$\\mu_{{SPE}}$& {:4.2e}\\\\', 0),)*, *histostyle='scatter'*, *datacolor='k'*, *modelcolor='r'*)
Show the fit result

Parameters

- **ymin** (*float*) – limit the yrange to ymin
- **xmax** (*float*) – limit the xrange to xmax
- **model_alpha** (*float*) – 0 <= x <= 1 the alpha value of the lineplot for the model
- **ylabel** (*str*) – label for yaxis
- **log** (*bool*) – plot in log scale
- **axes_range** (*str*) – the "field of view" to show
- **fig** (*pylab.figure*) – A figure instance
- **add_parameter_text** (*tuple*) – Display a parameter in the table on the plot ((text, parameter_number), (text, parameter_number),...)
- **datacolor** (*matplotlib color compatible*) –
- **modelcolor** (*matplotlib color compatible*) –

Returns pylab.figure

set_distribution(*distr*)
Assing a distribution to the model

Parameters **distr** –

Returns:

HErmes.fitting.model.**concat_functions**(*fncs*)
Inspect functions and construct a new one which returns the added result. concat_functions(A(x, apars), B(x, bpars)) -> C(x, apars,bpars) C(x, apars, bpars) returns (A(x, apars) + B(x, bpars))

Parameters **fncs** (*list*) – The callables to concat

Returns tuple (callable, list(pars))

HErmes.fitting.model.**construct_efunc**(*x*, *data*, *jointfunc*, *joint_pars*)
Construct a least-squares function

Parameters

- **x** –
- **data** –
- **jointfunc** –
- **joint_pars** –

Returns:

HErmes.fitting.model.**copy_func**(*f*)
Based on http://stackoverflow.com/a/6528148/190597 (Glenn Maynard)

HErmes.fitting.model.**create_minuit_pardict**(*fn*, *startparams*, *errors*, *limits*, *errordef*)
Construct a dictionary for minuit fitting

Parameters

- **fn** (*callable*) –
- **errors** (*list*) –
- **limits** (*list(tuple)*) –

Returns dict

Module contents

Simple to use fitting tools

HErmes.icecube_goodies package

Submodules

HErmes.icecube_goodies.conversions module

Unit conversions and such

HErmes.icecube_goodies.conversions.**ConvertPrimaryFromPDG**(*pid*)
Convert a primary id in an i3 file to the new values given by the pdg

HErmes.icecube_goodies.conversions.**ConvertPrimaryToPDG**(*pid*)
Convert a primary id in an i3 file to the new values given by the pdg

HErmes.icecube_goodies.conversions.**IsPDGEncoded**(*pid*, *neutrino=False*)
Check if the particle has already a pdg compatible pid

Parameters **id** (*int*) – Partilce Id

Keyword Arguments **neutrino** (*bool*) – as nue is H in PDG, set true if you know already that ihe particle might be a neutrino

Returns (bool): True if PDG compatible

class HErmes.icecube_goodies.conversions.**PDGCode**
Bases: object

Namespace for PDG conform particle type codes

Al26Nucleus = 1000130260

Al27Nucleus = 1000130270

Ar36Nucleus = 1000180360

Ar37Nucleus = 1000180370

Ar38Nucleus = 1000180380

Ar39Nucleus = 1000180390

Ar40Nucleus = 1000180400

```
Ar41Nucleus = 1000180410
Ar42Nucleus = 1000180420
B10Nucleus = 1000050100
B11Nucleus = 1000050110
Be9Nucleus = 1000040090
C12Nucleus = 1000060120
C13Nucleus = 1000060130
Ca40Nucleus = 1000200400
Ca41Nucleus = 1000200410
Ca42Nucleus = 1000200420
Ca43Nucleus = 1000200430
Ca44Nucleus = 1000200440
Ca45Nucleus = 1000200450
Ca46Nucleus = 1000200460
Ca47Nucleus = 1000200470
Ca48Nucleus = 1000200480
Cl35Nucleus = 1000170350
Cl36Nucleus = 1000170360
Cl37Nucleus = 1000170370
Cr50Nucleus = 1000240500
Cr51Nucleus = 1000240510
Cr52Nucleus = 1000240520
Cr53Nucleus = 1000240530
Cr54Nucleus = 1000240540
D0 = 421
D0Bar = -421
DMinus = -411
DPlus = 411
DsMinusBar = -431
DsPlus = 431
EMinus = 11
EPlus = -11
Eta = 221
F19Nucleus = 1000090190
Fe54Nucleus = 1000260540
Fe55Nucleus = 1000260550
```

```
Fe56Nucleus = 1000260560
Fe57Nucleus = 1000260570
Fe58Nucleus = 1000260580
Gamma = 22
He3Nucleus = 1000020030
He4Nucleus = 1000020040
K0_Long = 130
K0_Short = 310
K39Nucleus = 1000190390
K40Nucleus = 1000190400
K41Nucleus = 1000190410
KMinus = -321
KPlus = 321
Lambda = 3122
LambdaBar = -3122
LambdacPlus = 4122
Li6Nucleus = 1000030060
Li7Nucleus = 1000030070
Mg24Nucleus = 1000120240
Mg25Nucleus = 1000120250
Mg26Nucleus = 1000120260
Mn52Nucleus = 1000250520
Mn53Nucleus = 1000250530
Mn54Nucleus = 1000250540
Mn55Nucleus = 1000250550
MuMinus = 13
MuPlus = -13
N14Nucleus = 1000070140
N15Nucleus = 1000070150
Na23Nucleus = 1000110230
Ne20Nucleus = 1000100200
Ne21Nucleus = 1000100210
Ne22Nucleus = 1000100220
Neutron = 2112
NeutronBar = -2112
NuE = 12
```

```
NuEBar = -12
NuMu = 14
NuMuBar = -14
NuTau = 16
NuTauBar = -16
O16Nucleus = 1000080160
O17Nucleus = 1000080170
O18Nucleus = 1000080180
OmegaMinus = 3334
OmegaPlusBar = -3334
P31Nucleus = 1000150310
P32Nucleus = 1000150320
P33Nucleus = 1000150330
PMinus = -2212
PPlus = 2212
Pi0 = 111
PiMinus = -211
PiPlus = 211
S32Nucleus = 1000160320
S33Nucleus = 1000160330
S34Nucleus = 1000160340
S35Nucleus = 1000160350
S36Nucleus = 1000160360
Sc44Nucleus = 1000210440
Sc45Nucleus = 1000210450
Sc46Nucleus = 1000210460
Sc47Nucleus = 1000210470
Sc48Nucleus = 1000210480
Si28Nucleus = 1000140280
Si29Nucleus = 1000140290
Si30Nucleus = 1000140300
Si31Nucleus = 1000140310
Si32Nucleus = 1000140320
Sigma0 = 3212
Sigma0Bar = -3212
SigmaMinus = 3112
```

SigmaMinusBar = -3222

SigmaPlus = 3222

SigmaPlusBar = -3112

TauMinus = 15

TauPlus = -15

Ti44Nucleus = 1000220440

Ti45Nucleus = 1000220450

Ti46Nucleus = 1000220460

Ti47Nucleus = 1000220470

Ti48Nucleus = 1000220480

Ti49Nucleus = 1000220490

Ti50Nucleus = 1000220500

V48Nucleus = 1000230480

V49Nucleus = 1000230490

V50Nucleus = 1000230500

V51Nucleus = 1000230510

WMinus = -24

WPlus = 24

Xi0 = 3322

Xi0Bar = -3322

XiMinus = 3312

XiPlusBar = -3312

Z0 = 23

unknown = 0

class HErmes.icecube_goodies.conversions.**ParticleType**

Bases: object

Namespace for icecube particle type codes

Al26Nucleus = 2613

Al27Nucleus = 2713

Ar36Nucleus = 3618

Ar37Nucleus = 3718

Ar38Nucleus = 3818

Ar39Nucleus = 3918

Ar40Nucleus = 4018

Ar41Nucleus = 4118

Ar42Nucleus = 4118

```
B11Nucleus = 1105
Be9Nucleus = 904
C12Nucleus = 1206
Ca40Nucleus = 4020
Cl35Nucleus = 3517
Cr52Nucleus = 5224
EMinus = 3
EPlus = 2
F19Nucleus = 1909
Fe56Nucleus = 5626
Gamma = 1
He4Nucleus = 402
K0_Long = 10
K0_Short = 16
K39Nucleus = 3919
KMinus = 12
KPlus = 11
Li7Nucleus = 703
Mg24Nucleus = 2412
Mn55Nucleus = 5525
MuMinus = 6
MuPlus = 5
N14Nucleus = 1407
Na23Nucleus = 2311
Ne20Nucleus = 2010
Neutron = 13
NuE = 66
NuEBar = 67
NuMu = 68
NuMuBar = 69
NuTau = 133
NuTauBar = 134
O16Nucleus = 1608
P31Nucleus = 3115
PMinus = 15
PPlus = 14
```

Pi0 = 7

PiMinus = 9

PiPlus = 8

S32Nucleus = 3216

Sc45Nucleus = 4521

Si28Nucleus = 2814

TauMinus = 132

TauPlus = 131

Ti48Nucleus = 4822

V51Nucleus = 5123

unknown = 0

HErmes.icecube_goodies.fluxes module

Flux models for atmospheric neutrino and muon fluxes as well as power law fluxes

HErmes.icecube_goodies.fluxes.**AtmoWrap**(**args*, ***kwargs*)
Allows currying atmospheric flux functions for class interface :param *args: passed through to AtmosphericNuFlux :param **kwargs: passed through to AtmosphericNuFlux

Returns: AtmosphericNuFlux with applied arguments

HErmes.icecube_goodies.fluxes.**AtmosphericNuFlux**(**args*, ***kwargs*)

class HErmes.icecube_goodies.fluxes.**ICMuFluxes**
Bases: object

GaisserH3a = None

GaisserH4a = None

Hoerandel = None

Hoerandel5 = None

class HErmes.icecube_goodies.fluxes.**MuFluxes**
Bases: object

Namespace for atmospheric muon fluxes

GaisserH3a = None

GaisserH4a = None

Hoerandel = None

Hoerandel5 = None

class HErmes.icecube_goodies.fluxes.**NuFluxes**
Bases: object

Namespace for neutrino fluxes

static BARTOL(*x*)

static BERSSH3a(*x*)

static BERSSH4a(*x*)

static E2(*mc_p_energy*, *mc_p_type*, *mc_p_zenith*, *fluxconst=1e-08*, *gamma=-2*)

static ERS(*x*)

static ERSH3a(*x*)

static ERSH4a(*x*)

static Honda2006(*x*)

static Honda2006H3a(*x*)

static Honda2006H4a(*x*)

HErmes.icecube_goodies.fluxes.**PowerLawFlux**(*fluxconst=1e-08*, *gamma=2*)
A simple powerlaw flux

Parameters

- **fluxconst** (*float*) – normalization
- **gamma** (*float*) – spectral index

Returns (func): the flux function

HErmes.icecube_goodies.fluxes.**PowerWrap**(**args*, ***kwargs*)
Allows currying PowerLawFlux for class interface

Parameters

- ***args** – applied to PowerLawFlux
- ****kwargs** – applied to PowerLawFlux

Returns: PowerLawFlux with applied arguments

HErmes.icecube_goodies.fluxes.**generated_corsika_flux**(*ebinc*, *datasets*)
Calculate the livetime of a number of given coriska datasets using the weighting moduel The calculation here means a comparison of the number of produced events per energy bin with the expected event yield from fluxes in nature. If necessary call home to the simprod db. Works for 5C datasets.

Parameters

- **ebinc** (*np.array*) – Energy bins (centers)
- **datasets** (*list*) – A list of dictionaries with properties of the datasets or dataset numbers. If only nu8mbers are given, then simprod db will be queried format of dataset dict: example_datasets ={42: {"nevents": 1, "nfiles": 1, "emin": 1, "emax": 1, "normalization": [10., 5., 3., 2., 1.], "gamma": [-2.]*5, "LowerCutoffType": 'EnergyPerNucleon', "UpperCutoffType": 'EnergyPerParticle', "height": 1600, "radius": 800}}

Returns tuple (generated protons, generated irons)

HErmes.icecube_goodies.helpers module

Goodies for icecube

class HErmes.icecube_goodies.helpers.**IceCubeGeometry**
Bases: object

Provide icecube geometry information

coordinates(*string*, *dom*)
Calculate the xy position of a given string

load_geo()
Load geometry information

HErmes.icecube_goodies.weighting module

An interface to icecube's weighting schmagoigl

HErmes.icecube_goodies.weighting.**GetGenerator**(*datasets*)
datasets must be a dict of dataset_id : number_of_files

Parameters **datasets** (*dict*) – Query the database for these datasets. dict dataset_id -> number of files

Returns (icecube.weighting...): Generation probability object

HErmes.icecube_goodies.weighting.**GetModelWeight**(*model*, *datasets*, *mc_datasets=None*, *mc_p_en=None*, *mc_p_ty=None*, *mc_p_ze=None*, *mc_p_we=1.0*, *mc_p_ts=1.0*, *mc_p_gw=1.0*, ***model_kwargs*)
Compute weights using a predefined model

Parameters

- **model** (*func*) – Used to calculate the target flux
- **datasets** (*dict*) – Get the generation pdf for these datasets from the db dict needs to be dataset_id -> nfiles

Keyword Arguments

- **mc_p_en** (*array-like*) – primary energy
- **mc_p_ty** (*array-like*) – primary particle type
- **mc_p_ze** (*array-like*) – primary particle cos(zenith)
- **mc_p_we** (*array-like*) – weight for mc primary, e.g. some interaction probability

Returns (array-like): Weights

class HErmes.icecube_goodies.weighting.**Weight**(*generator*, *flux*)
Bases: object

Provides the weights for weighted MC simulation. Uses the pdf from simulation and the desired flux

HErmes.icecube_goodies.weighting.**constant_weights**(*size*, *scale=1.0*)
Calculate a constant weight for all the entries, e.g. unity

Parameters **size** (*int*) – The size of the returned arraz (d)

Keyword Arguments **scale** (*float*) – The returned weight is 1/scale

Returns np.ndarray

HErmes.icecube_goodies.weighting.**get_weight_from_weightmap**(*model*, *datasets*, *mc_datasets=None*, *mc_p_en=None*, *mc_p_ty=None*, *mc_p_ze=None*, *mc_p_we=1.0*, *mc_p_ts=1.0*, *mc_p_gw=1.0*, ***model_kwargs*)

Get weights for weighted datasets (generation spectra is already the target flux)

Parameters

- **model** (*func*) – Not used, only for compatibility
- **datasets** (*dict*) – used to provide nfiles

Keyword Arguments

- **mc_p_en** (*array-like*) – primary energy
- **mc_p_ty** (*array-like*) – primary particle type
- **mc_p_ze** (*array-like*) – primary particle cos(zenith)
- **mc_p_we** (*array-like*) – weight for mc primary, e.g. some interaction probability
- **mc_p_gw** (*array-like*) – generation weight
- **mc_p_ts** (*array-like*) – mc timescale
- **mc_datasets** (*array-like*) – an array which has per-event dataset information

Returns (array-like): Weights

Module contents

HErmes.plotting package

Submodules

HErmes.plotting.canvases module

Provides canvases for multi axes plots

HErmes.plotting.canvases.**Image**(*x*)

class HErmes.plotting.canvases.**YStackedCanvas**(*subplot_yheights=(0.2, 0.2, 0.5)*, *padding=(0.15, 0.05, 0.0, 0.1)*, *space_between_plots=0*, *figsize='auto'*, *figure_factory=None*)

Bases: object

A canvas for plotting multiple axes

eliminate_lower_yticks()

Eliminate the lowest y tick on each axes. The bottom axes keeps its lowest y-tick.

global_legend(**args*, ***kwargs*)

A combined legend for all axes

Parameters args will be passed to pylab.legend(*all*) –

Keyword Arguments **kwargs will be passed to pylab.legend**(*all*) –

limit_xrange(*xmin=None*, *xmax=None*)
Walk through all axes and set xlims

Keyword Arguments

- **xmin** (*float*) – left x edge of axes
- **xmax** (*float*) – right x edge of axes

Returns None

limit_yrange(*ymin=None*, *ymax=None*)
Walk through all axes and adjust ymin and ymax

Keyword Arguments **ymin** (*float*) – min ymin value

save(*path*, *name*, *formats=('pdf', 'png')*, ***kwargs*)
Calls pylab.savefig for all endings

Parameters

- **path** (*str*) – path to savefile
- **name** (*str*) – filename to save
- **formats** (*tuple*) – for each name in endings, a file is save

Keyword Arguments **keyword args will be passed to pylab.savefig**(*all*) –

Returns The full path to the the saved file

Return type str

select_axes(*axes*)
Set the scope on a certain axes

Parameters **axes** (*int*) – 0 lowest, -1 highest, increasing y-order

Returns The axes instance

Return type matplotlib.axes.axes

show()
Use the IPython.core.Image to show the plot

Returns the plot

Return type IPython.core.Image

HErmes.plotting.colors module

Color management - provide a nice color scheme even if seaborn is not available

class HErmes.plotting.colors.**ColorDict**
Bases: dict

Use to transparently map any non defined colors through, like string like 'k' for matplotlib

HErmes.plotting.colors.**get_color_palette**(*name='dark'*)
Load a color pallete, use seaborn if available.

Keyword Arguments **name** (*str*) – A string which is passed though seaborn.color_palette

Returns HErmes.plotting.plot_colors.ColorDict

HErmes.plotting.layout module

A set of figure sizes for publication-ready figures on A4 paper based on the golden ratio. For the use with pylabe figure, e.g. fig =pylab.figure(figsize=FIGSIZE_A4)

Available layouts:

- FIGSIZE_A4: Figure on full A4 width, portrait mode, golden ratio
- FIGSIZE_A4_LANDSCAPE: Figure on full A4 width, landscape mode, golden ratio
- FIGSIZE_A4_LANDSCAPE_HALF_HEIGHT: Figure on full A4 width, landscape mode, height half of golden ratio
- FIGSIZE_A4_SQUARE: Figure on full A4 width, square

HErmes.plotting.plotting module

Define some

class HErmes.plotting.plotting.**VariableDistributionPlot**(*cuts=None*, *color_palette='dark'*, *bins=None*, *xlabel=None*)

Bases: object

A plot which shows the distribution of a certain variable. Cuts can be indicated with lines and arrows. This class defines (and somehow enforces) a certain style.

add_cumul(*name*)
Add a cumulative distribution to the plto

Parameters **name** (*str*) – the name of the category

add_cuts(*cut*)
Add a cut to the the plot which can be indicated by an arrow

Parameters **cuts** (*HErmes.selection.cuts.Cut*) –

Returns None

add_data(*variable_data*, *name*, *bins=None*, *weights=None*, *label=''*)
Histogram the added data and store internally

Parameters

- **name** (*string*) – the name of a category
- **variable_data** (*array*) – the actual data

Keyword Arguments

- **bins** (*array*) – histogram binning
- **weights** (*array*) – weights for the histogram
- **label** (*str*) – A label for the data when plotted

add_legend(***kwargs*)
Add a legend to the plot. If no kwargs are passed, use some reasonable default.

Keyword Arguments be passed to pylab.legend (*will*) –

add_ratio(*nominator*, *denominator*, *total_ratio=None*, *total_ratio_errors=None*, *log=False*, *label='data/$\\Sigma$ bg'*)
Add a ratio plot to the canvas

Parameters

- **nominator** (*list or str*) – name(s) of the categorie(s) which will be the nominator in the ratio
- **denominator** (*list or str*) – name(s) of the categorie(s) which will be the nominator in the ratio

Keyword Arguments

- **total_ratio** (*bool*) – Indicate the total ratio with a line in the plot
- **total_ratio_errors** (*bool*) – Draw error region around total ratio
- **log** (*bool*) – draw ratio plot in log-scale
- **label** (*str*) – y-label for the ratio plot

add_variable(*category*, *variable_name*, *external_weights=None*, *transform=None*)
Convenience interface if data is sorted in categories already

Parameters

- **category** (*HErmese.variables.category.Category*) – Get variable from this category
- **variable_name** (*string*) – The name of the variable

Keyword Arguments

- **external_weights** (*np.ndarray*) – Supply an array for weighting. This will OVERIDE ANY INTERNAL WEIGHTING MECHANISM and use the supplied weights.
- **transform** (*callable*) – Apply transformation todata

indicate_cut(*ax*, *arrow=True*)
If cuts are given, indicate them by lines

Parameters **ax** (*pylab.axes*) – axes to draw on

static optimal_plotrange_histo(*histograms*)
Get most suitable x and y limits for a bunc of histograms

Parameters **histograms** (*list(d.factory.hist1d)*) – The histograms in question

Returns xmin, xmax, ymin, ymax

Return type tuple (float, float, float, float)

plot(*axes_locator=((0, 'c', 0.2), (1, 'r', 0.2), (2, 'h', 0.5))*, *combined_distro=True*, *combined_ratio=True*, *combined_cumul=True*, *normalized=True*, *log=True*, *legendwidth=1.5*, *ylabel='rate/bin [1/s]'*, *figure_factory=None*)
Create the plot

Keyword Arguments

- **heights** –
- **axes_locator** –
- **combined_distro** –
- **combined_ratio** –

- **combined_cumul** –
- **log** –
- **normalized** (*bool*) –

Returns:

HErmes.plotting.plotting.**adjust_minor_ticks**(*axis*, *which='x'*)

Decorate the x-axis with a reasonable set of minor x-ticks

Parameters **axis** (*matplotlib.axis*) – The axis to decorate

Keyword Arguments **which** (*str*) – either "x", "y" or "both"

Returns matplotlib.axis

HErmes.plotting.plotting.**create_arrow**(*ax*, *x_0*, *y_0*, *dx*, *dy*, *length*, *width=0.1*, *shape='right'*, *fc='k'*, *ec='k'*, *alpha=1.0*, *log=False*)

Create an arrow object for plots. This is typically a large arrow, which can used to indicate a region in the plot which is excluded by a cut.

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – The axes where the arrow will be attached to
- **x_0** (*float*) – x-origin of the arrow
- **y_0** (*float*) – y-origin of the arrow
- **dx** (*float*) – x length of the arrow
- **dy** (*float*) – y length of the arrow
- **length** (*float*) – additional scaling parameter to scale the length of the arrow

Keyword Arguments

- **width** (*float*) – thickness of arrow
- **shape** (*str*) – either "full", "left" or "right"
- **fc** (*str*) – facecolor
- **ec** (*str*) – edgecolor
- **alpha** (*float*) – 0 -1 alpha value of the arrow
- **log** (*bool*) – I for logscale, the proportions of the arrow will be adjusted accorginly.

Returns matplotlib.axes._subplots.AxesSubplot

HErmes.plotting.plotting.**line_plot**(*quantities*, *bins=None*, *xlabel=''*, *add_ratio=None*, *ratiolabel=''*, *colors=None*, *figure_factory=None*)

Parameters **quantities** –

Keyword Arguments

- **bins** –
- **xlabel** –
- **add_ratio** (*tuple*) – (["data1"],["data2"])
- **ratiolabel** (*str*) –
- **colors** –

- **figure_factory** (*callable*) – Factory function returning matpltolib.Figure

Returns:

HErmes.plotting.plotting.**meshgrid**(*xs*, *ys*)

Create x and y data for matplotlib pcolormesh and similar plotting functions.

Parameters

- **xs** (*np.ndarray*) – 1d x bins
- **ys** (*np.ndarray*) – 2d y bins

Returns 2d X and 2d Y matrices as well as a placeholder for the Z array

Return type tuple (np.ndarray, np.ndarray, np.ndarray)

Module contents

A set of

HErmes.plotting.**add_styles**()

HErmes.selection package

Submodules

HErmes.selection.categories module

Categories of data, like "signal" of "background" etc

class HErmes.selection.categories.**AbstractBaseCategory**(*name*)

Bases: object

Stands for a specific type of data, e.g. detector data in a specific configuarion, simulated data etc.

add_cut(*cut*)

Add a cut without applying it yet

Parameters **cut** (*pyevsel.variables.cut.Cut*) – Append this cut to the internal cutlist

add_livetime_weighted(*other*, *self_livetime=None*, *other_livetime=None*)

Combine two datasets livetime weighted. If it is simulated data, then in general it does not know about the detector livetime. In this case the livetimes for the two datasets can be given

Parameters **other** (*pyevsel.categories.Category*) – Add this dataset

Keyword Arguments

- **self_livetime** (*float*) – the data livetime for this dataset
- **other_livetime** (*float*) – the data livetime for the other dataset

add_plotoptions(*options*)

Add options on how to plot this category. If available, they will be used.

Parameters **options** (*dict*) – For the names which are currently supported, please see the example file

add_variable(*variable*)

Add a variable to this category

Parameters **variable** (*pyevsel.variables.variables.Variable*) – A Variable instalce

apply_cuts(*inplace=False*)
Apply the added cuts.

Keyword Arguments **inplace** (*bool*) – If True, cut the internal variable buffer (Can not be undone except variable is reloaded)

calculate_weights(*model*, *model_args=None*)

declare_harvested()
Set the flag that all the variables have been read out

delete_cuts()
Get rid of previously added cuts and undo them

delete_variable(*varname*)
Remove a variable entirely from the category

Parameters **varname** (*str*) – The name of the variable as stored in self.variable dict

Returns None

distribution(*varname*, *bins=None*, *color=None*, *alpha=0.5*, *fig=None*, *xlabel=None*, *norm=False*, *filled=None*, *legend=True*, *style='line'*, *log=False*, *transform=None*, *extra_weights=None*, *figure_factory=None*, *return_histo=False*)
Plot the distribution of variable in the category

Parameters **varname** (*str*) – The name of the variable in the catagory

Keyword Arguments

- **bins** (*int/np.ndarray*) – Bins for the distribution
- **color** (*str/int*) – A color identifier, either number 0-5 or matplotlib compatible
- **alpha** (*float*) – 0-1 alpha value for histogram
- **fig** (*matplotlib.figure.Figure*) – Canvas for plotting, if None an empty one will be created
- **xlabel** (*str*) – xlabel for the plot. If None, default is used
- **norm** (*str*) – "n" or "density" - make normed histogram
- **style** (*str*) – Either "line" or "scatter"
- **filled** (*bool*) – Draw filled histogram
- **legend** (*bool*) – if available, plot a legend
- **transform** (*callable*) – Apply transformation to the data before plotting
- **log** (*bool*) – Plot yaxis in log scale
- **extra_weights** (*numpy.ndarray*) – Use this for weighting. Will overwrite any other weights in the dataset
- **figure_factory** (*func*) – Must return a single matplotlib.Figure, NOTE: figure_factory has priority over fig keyword
- **return_histo** (*bool*) – Return the histogram instead of the figure. WARNING: changes return type!

Returns matplotlib.figure.Figure or dashi.histogram.hist1d

distribution2d(*varnames*, *bins=None*, *figure_factory=None*, *fig=None*, *norm=False*, *log=True*, *cmap=<Mock name='mock.get_cmap()' id='139867043115248'>*, *interpolation='gaussian'*, *cblabel='events'*, *weights=None*, *despine=False*, *return_histo=False*)

Draw a 2d distribution of 2 variables in the same category. :param varnames: The names of the variable in the catagory :type varnames: tuple(str,str)

Keyword Arguments

- **bins** (*tuple(int/np.ndarray)*) – Bins for the distribution
- **cmap** – A colormap
- **alpha** (*//*) – 0-1 alpha value for histogram
- **fig** (*matplotlib.figure.Figure*) – Canvas for plotting, if None an empty one will be created
- **xlabel** (*//*) – xlabel for the plot. If None, default is used
- **norm** (*str*) – "n" or "density" - make normed histogram
- **style** (*//*) – Either "line" or "scatter"
- **transform** (*callable*) – Apply transformation to the data before plotting
- **log** (*bool*) – Plot yaxis in log scale
- **figure_factory** (*func*) – Must return a single matplotlib.Figure, NOTE: figure_factory has priority over fig keyword
- **return_histo** (*bool*) – Return the histogram instead of the figure. WARNING: changes return type!

Returns matplotlib.figure.Figure or dashi.histogram.hist1d

drop_empty_variables()

Delete variables which have no len

Returns None

explore_files()

Get a sneak preview of what variables are avaukabke for readout

Returns list

get(*varkey*, *uncut=False*)

Retrieve the data of a variable

Parameters **varkey** (*str*) – The name of the variable

Keyword Arguments **uncut** (*bool*) – never return cutted values

get_datacube()

get_files(**args*, ***kwargs*)

Load files for this category uses HErmes.utils.files.harvest_files

Parameters ***args** (*list of strings*) – Path to possible files

Keyword Arguments

- **(dict(dataset_id** (datasets) – nfiles)): i given, load only files from dataset dataset_id set nfiles parameter to amount of L2 files the loaded files will represent
- **force** (*bool*) – forcibly reload filelist (pre-readout vars will be lost)
- **append** (*bool*) – keep the already aquired files and only append the new ones

- **other kwargs will be passed to** (*all*) –
- **utils.files.harvest_files** –

harvested

integrated_rate
Calculate the total eventrate of this category (requires weights)

Returns (tuple): rate and quadratic error

load_vardefs (*module*)
Load the variable definitions from a module

Parameters **module** (*python module*) – Needs to contain variable definitions

raw_count
Gives a number of "how many events are actually there"

Returns int

read_variables (*names=None*, *max_cpu_cores=6*)
Harvest the variables in self.vardict

Keyword Arguments

- **names** (*list*) – havest only these variables
- **max_cpu_cores** (*list*) – use a maximum of X cores of the cpu

show ()
Print out the names of the loaded variables

Returns dict (name, len)

undo_cuts ()
Conveniently undo a previous "apply_cuts"

variablenames

weights

weightvarname = None

class HErmes.selection.categories.**CombinedCategory** (*name*, *categories*)
Bases: object

Create a combined category out of several others This is mainly useful for plotting FIXME: should this inherit from category as well? The difference compared to the dataset is that this is flat

add_plotoptions (*options*)
Add options on how to plot this category. If available, they will be used.

Parameters **options** (*dict*) – For the names which are currently supported, please see the example file

get (*varname*)

integrated_rate
Calculate the total eventrate of this category (requires weights)

Returns (tuple): rate and quadratic error

vardict

weights

class HErmes.selection.categories.**Data**(*name*)
Bases: *HErmes.selection.categories.AbstractBaseCategory*

An interface to real time event data Simplified weighting only

calculate_weights(*model=None*, *model_args=None*)
Calculate weights as rate, that is number of events per livetime

Keyword Args: for compatibility...

estimate_livetime(*force=False*)
Calculate the livetime from run start/stop times, account for gaps

Keyword Arguments **force** (*bool*) – overide existing livetime

livetime

set_livetime(*livetime*)
Override the private _livetime member

Parameters **livetime** – The time needed for data-taking

Returns None

set_run_start_stop(*runstart_var=<Variable: None>*, *runstop_var=<Variable: None>*)
Let the simulation category know which are the paramters describing the primary

Keyword Arguments

- **runstart_var** (*pyevself.variables.variables.Variable/str*) – beginning of a run
- **runstop_var** (*pyevself.variables.variables.Variable/str*) – beginning of a run

set_weightfunction(*func*)

class HErmes.selection.categories.**ReweightedSimulation**(*name*, *mother*)
Bases: *HErmes.selection.categories.Simulation*

A proxy for simulation dataset, when only the weighting differs

add_livetime_weighted(*other*)
Combine two datasets livetime weighted. If it is simulated data, then in general it does not know about the detector livetime. In this case the livetimes for the two datasets can be given

Parameters **other** (*pyevsel.categories.Category*) – Add this dataset

Keyword Arguments

- **self_livetime** (*float*) – the data livetime for this dataset
- **other_livetime** (*float*) – the data livetime for the other dataset

datasets

files

get(*varname*, *uncut=False*)
Retrieve the data of a variable

Parameters **varkey** (*str*) – The name of the variable

Keyword Arguments **uncut** (*bool*) – never return cutted values

harvested

mother

raw_count
Gives a number of "how many events are actually there"

Returns int

read_mc_primary(*energy_var='mc_p_en'*, *type_var='mc_p_ty'*, *zenith_var='mc_p_ze'*, *weight_var='mc_p_we'*)
Trigger the readout of MC Primary information Rename variables to magic keywords if necessary

Keyword Arguments

- **energy_var** (*str*) – simulated primary energy
- **type_var** (*str*) – simulated primary type
- **zenith_var** (*str*) – simulated primary zenith
- **weight_var** (*str*) – a weight, e.g. interaction propability

read_variables(*names=None*, *max_cpu_cores=6*)
Harvest the variables in self.vardict

Keyword Arguments

- **names** (*list*) – havest only these variables
- **max_cpu_cores** (*list*) – use a maximum of X cores of the cpu

setter(*other*)

vardict

class HErmes.selection.categories.**Simulation**(*name*, *weightvarname=None*)
Bases: *HErmes.selection.categories.AbstractBaseCategory*

An interface to variables from simulated data Allows to weight the events

calculate_weights(*model=None*, *model_args=None*)
Walk the variables of this category and identify the weighting variables and calculate them.

Usage example: calculate_weights(model=lambda x: np.pow(x, -2.), model_args=["primary_energy"])

Keyword Arguments

- **model** (*func*) – The target flux to weight to, if None, generated flux is used for weighting
- **model_args** (*list*) – The variables the model should be applied to from the variable dict

Returns np.ndarray

livetime

mc_p_readout

read_mc_primary(*energy_var='mc_p_en'*, *type_var='mc_p_ty'*, *zenith_var='mc_p_ze'*, *weight_var='mc_p_we'*)
Trigger the readout of MC Primary information Rename variables to magic keywords if necessary

Keyword Arguments

- **energy_var** (*str*) – simulated primary energy
- **type_var** (*str*) – simulated primary type
- **zenith_var** (*str*) – simulated primary zenith
- **weight_var** (*str*) – a weight, e.g. interaction propability

HErmes.selection.categories.**cut_with_nans**(*data*, *cutmask*)
Cut the individual fields of a 2d array and keep the shape by filling up with nans

Parameters

- **data** (*np.ndarray*) – The array to cut
- **cutmask** (*np.ndarray*) – Cut with this boolean array

Returns data with applied cuts

Return type np.ndarray

HErmes.selection.cut module

Remove part of the data which falls below a certain criteria.

class HErmes.selection.cut.**Cut**(**cuts*, ***kwargs*)
Bases: object

Cuts are basically conditions on a set of parameters.

variablenames
The names of the variables the cut will be applied to

HErmes.selection.dataset module

Datasets group categories together. Method calls on datasets invoke the individual methods on the individual categories. Cuts applied to datasets will act on each individual category.

class HErmes.selection.dataset.**Dataset**(**args*, ***kwargs*)
Bases: object

Holds different categories, relays calls to each of them.

add_category(*category*)
Add another category to the dataset

Parameters **category** (*HErmes.selection.categories.Category*) – add this category

add_cut(*cut*)
Add a cut without applying it yet

Parameters **cut** (*HErmes.selection.variables.cut.Cut*) – Append this cut to the internal cutlist

add_variable(*variable*)
Add a variable to this category

Parameters **variable** (*HErmes.selection.variables.variables.Variable*) – A Variable instalce

apply_cuts(*inplace=False*)
Apply them all!

calc_ratio(*nominator=None*, *denominator=None*)
Calculate a ratio of the given categories

Parameters

- **nominator** (*list*) –

- **denominator** (*list*) –

Returns tuple

calculate_weights(*model=None*, *model_args=None*)
Calculate the weights for all categories

Keyword Arguments

- **model** (*dict/func*) – Either a dict catname -> func or a single func If it is a single funct it will be applied to all categories
- **model_args** (*dict/list*) – variable names as arguments for the function

categorynames

combined_categorynames

delete_cuts()
Completely purge all cuts from this dataset

delete_variable(*varname*)
Delete a variable entirely from the dataset

Parameters **varname** (*str*) – the name of the variable

Returns None

distribution(*name*, *ratio=([]*[ ], *)*, *cumulative=True*, *log=False*, *transform=None*, *color_palette='dark'*, *normalized=False*, *styles={}*, *style='classic'*, *ylabel='rate/bin [1/s]'*, *axis_properties=None*, *ratiolabel='data/$\\Sigma$ bg'*, *bins=None*, *external_weights=None*, *figure_factory=None*)
One shot short-cut for one of the most used plots in eventselections

Parameters **name** (*string*) – The name of the variable to plot

Keyword Arguments

- **path** (*str*) – The path under which the plot will be saved.
- **ratio** (*list*) – A ratio plot of these categories will be crated
- **color_palette** (*str*) – A predifined color palette (from seaborn or HErmes.plotting.colors)
- **normalized** (*bool*) – Normalize the histogram by number of events
- **transform** (*callable*) – Apply this transformation before plotting
- **styles** (*dict*) – plot styling options
- **ylabel** (*str*) – general label for y-axis
- **ratiolabel** (*str*) – different label for the ratio part of the plot
- **bins** (*np.ndarray*) – binning, if None binning will be deduced from the variable definition
- **figure_factory** (*func*) – factory function which return a matplotlib.Figure
- **style** (*string*) – TODO “modern” || “classic” || “modern-cumul” || “classic-cumul”
- **external_weights** (*dict*) – supply external weights - this will OVERIDE ANY INTERNALLY CALCULATED WEIGHTS and use the supplied weights instead. must be in the form { “categoryname” : weights}
- **axis_properties** (*dict*) – Manually define a plot layout with up to three axes. For example, it can look like this: {

"top": {"type": "h", # histogram "height": 0.4, # height in percent "index": 2}, # used internally

"center": {"type": "r", # ratio plot "height": 0.2, "index": 1},

"bottom": { "type": "c", # cumulative histogram "height": 0.2, "index": 0}

}

Returns HErmes.selection.variables.VariableDistributionPlot

drop_empty_variables()
Delete variables which have no len

Returns None

files

get_category(*categoryname*)
Get a reference to a category.

Parameters **category** – A name which has to be associated to a category

Returns HErmes.selection.categories.Category

get_sparsest_category(*omit_empty_cat=True*)
Find out which category of the dataset has the least statistical power

Keyword Arguments **omit_empty_cat** (*bool*) – if a category has no entries at all, omit

Returns category name

Return type str

get_variable(*varname*)
Get a pandas dataframe for all categories

Parameters **varname** (*str*) – A name of a variable

Returns A 2d dataframe category -> variable

Return type pandas.DataFrame

integrated_rate
Integrated rate for each category

Returns rate with error

Return type pandas.Panel

load_vardefs(*vardefs*)
Load the variable definitions from a module

Parameters **vardefs** (*python module/dict*) – A module needs to contain variable definitions. It can also be a dictionary of categoryname->module

read_variables(*names=None*, *max_cpu_cores=6*)
Read out the variable for all categories

Keyword Arguments

- **names** (*str*) – Readout only these variables if given
- **max_cpu_cores** (*int*) – Maximum number of cpu cores which will be used

Returns None

set_default_plotstyles(*styledict*)
Define a standard for each category how it should appear in plots

Parameters **styledict** (*dict*) –

set_livetime(*livetime*)
Define a livetime for this dataset.

Parameters **livetime** (*float*) – Time interval the data was taken in. (Used for rate calculation)

Returns None

set_weightfunction(*weightfunction=<function Dataset.<lambda>>*)
Defines a function which is used for weighting

Parameters **weightfunction** (*func or dict*) – if func is provided, set this to all categories if needed, provide dict, cat.name -> func for individula setting

Returns None

sum_rate(*categories=None*)
Sum up the integrated rates for categories

Parameters **categories** – categories considerred background

Returns rate with error

Return type tuple

tinytable(*signal=None*, *background=None*, *layout='v'*, *format='html'*, *order_by=<function Dataset.<lambda>>*, *livetime=1.0*)
Use dashi.tinytable.TinyTable to render a nice html representation of a rate table

Parameters

- **signal** (*list*) – summing up signal categories to calculate total signal rate
- **background** (*list*) – summing up background categories to calculate total background rate
- **layout** (*str*) – "v" for vertical, "h" for horizontal
- **format** (*str*) – "html","latex","wiki"

Returns formatted table in desired markup

Return type str

undo_cuts()
Undo previously done cuts, but keep them so that they can be re-applied

variablenames

weights
Get the weights for all categories in this dataset

HErmes.selection.dataset.**get_label**(*category*)
Get the label for labeling plots from a datasets plot_options dictionary.

Parameters **category** (*HErmes.selection.categories.category*) – Query the category's plot_options dict, if not fall back to category.name

Returns string

HErmes.selection.magic_keywords module

All magic keywords shall summon here

HErmes.selection.variables module

Container classes for variables

class HErmes.selection.variables.**AbstractBaseVariable**
Bases: object

Read out tagged numerical data from files

ROLES
alias of *VariableRole*

bins

calculate_fd_bins()
Calculate a reasonable binning

Returns Freedman Diaconis bins

Return type numpy.ndarray

data

declare_harvested()

harvest(**files*)
Hook to the harvest method. Don't use in case of multiprocessing! :param *files: walk through these files and readout

harvested

ndim

rewire_variables(*vardict*)

class HErmes.selection.variables.**CompoundVariable**(*name*, *variables=None*, *label=''*, *bins=None*, *operation=<function CompoundVariable.<lambda>>*, *role=<VariableRole.SCALAR: 10>*)
Bases: *HErmes.selection.variables.AbstractBaseVariable*

Calculate a variable from other variables. This kind of variable will not read any file.

harvest(**filenames*)
Hook to the harvest method. Don't use in case of multiprocessing! :param *files: walk through these files and readout

rewire_variables(*vardict*)
Use to avoid the necessity to read out variables twice as the variables are copied over by the categories, the refernce is lost. Can be rewired though

class HErmes.selection.variables.**Variable**(*name*, *definitions=None*, *bins=None*, *label=''*, *transform=<function Variable.<lambda>>*, *role=<VariableRole.SCALAR: 10>*, *nevents=None*, *reduce_dimension=None*)
Bases: *HErmes.selection.variables.AbstractBaseVariable*

A hook to a single variable read out from a file

rewire_variables(*vardict*)
Make sure all the variables are connected properly. This is only needed for combined/compound variables

Returns None

class HErmes.selection.variables.**VariableList**(*name*, *variables=None*, *label=''*, *bins=None*, *role=<VariableRole.SCALAR: 10>*)
Bases: *HErmes.selection.variables.AbstractBaseVariable*

A list of variable. Can not be read out from files.

data

harvest(**filenames*)
Hook to the harvest method. Don't use in case of multiprocessing! :param *files: walk through these files and readout

rewire_variables(*vardict*)
Use to avoid the necessity to read out variables twice as the variables are copied over by the categories, the refernce is lost. Can be rewired though

class HErmes.selection.variables.**VariableRole**
Bases: enum.Enum

Define roles for variables. Some variables used in a special context (like weights) are easily recognizable by this flag.

ARRAY = 20

ENDTIME = 70

EVENTID = 50

GENERATORWEIGHT = 30

RUNID = 40

SCALAR = 10

STARTIME = 60

UNKNOWN = 0

HErmes.selection.variables.**extract_from_root**(*filename*, *definitions*, *nevents=None*, *reduce_dimension=None*)
Use the uproot system to get information from rootfiles. Supports a basic tree of primitive datatype like structure.

Parameters

- **filename** (*str*) – datafile
- **defininitiions** (*list*) – tree and branch adresses

Keyword Arguments

- **nevents** (*int*) – number of events to read out
- **reduce_dimension** (*int*) – If data is vector-type, reduce it by taking the n-th element

HErmes.selection.variables.**freedman_diaconis_bins**(*data*, *leftedge*, *rightedge*, *minbins=20*, *maxbins=70*, *fallbackbins=70*)
Get a number of bins for a histogram following Freedman/Diaconis

Parameters

- **leftedge** (*float*) – left bin edge
- **rightedge** (*float*) – right bin edge
- **minbins** (*int*) – the minimum number of bins
- **maxbins** (*int*) – the maximum number of bins
- **fallbackbins** (*int*) – a number of bins which is returned if calculation failse

Returns number of bins, minbins < bins < maxbins

Return type nbins (int)

HErmes.selection.variables.**harvest**(*filenames*, *definitions*, ***kwargs*)

Read variables from files into memory. Will be used by HErmes.selection.variables.Variable.harvest This will be run multi-threaded. Keep that in mind, arguments have to be picklable, also everything thing which is read out must be picklable. Lambda functions are NOT picklable

Parameters

- **filenames** (*list*) – the files to extract the variables from. currently supported: hdf
- **definitions** (*list*) – where to find the data in the files. They usually have some tree-like structure, so this a list of leaf-value pairs. If there is more than one all of them will be tried. (As it might be that in some files a different naming scheme was used) Example: [("hello_reoncstruction", "x"), ("hello_reoncstruction", "y")]]

Keyword Arguments

- **transformation** (*func*) – After the data is read out from the files, transformation will be applied, e.g. the log to the energy.
- **fill_empty** (*bool*) – Fill empty fields with zeros
- **nevents** (*int*) – ROOT only - read out only nevents from the files
- **reduce_dimension** (*str*) – ROOT only - multidimensional data can be reduced by only using the index given by reduce_dimension. E.g. in case of a TVector3, and we want to have onlz x, that would be 0, y -> 1 and z -> 2.
- **FIXME** – Not implemented yet! precision (int): Precision in bit

Returns pd.Series or pd.DataFrame

Module contents

Provides containers for in-memory variable. These containers are called "categroies", and they represent a set of variables for a certain type of data. Categories can be further grouped into "Datasets". Variables can be read out from files and stored in memory in the form of numpy arrays or pandas DataSeries/DataFrames. Selection criteria can be applied simultaniously (and reversibly) to all categories in a dataset with the "Cut" class.

HErmes.selection provides the following submodules:

- *categories* : Container classes for variables.
- *dataset* : Grouping categories together.
- *cut* : Apply selection criteria on variables in a category.
- *variables* : Variable definition. Harvest variables from files.
- *magic_keywords* : A bunch of fixed names for automatic weight calculation.

HErmes.selection.**load_dataset**(*config*, *variables=None*, *max_cpu_cores=6*)
Read a json configuration file and load a dataset populated with variables from the files given in the configuration file.

Parameters **config** (*str/dict*) – json style config file or dict

Keyword Arguments

- **variables** (*list*) – list of strings of variable names to read out
- **max_cpu_cores** (*int*) – maximum number of cpu ucores to use for variable readout

Returns HErmes.selection.dataset.Dataset

HErmes.utils package

Submodules

HErmes.utils.files module

Locate files on the filesystem and group them together

HErmes.utils.files.**DS_ID**(*filename*)

HErmes.utils.files.**ENDING**(*filename*)

HErmes.utils.files.**EXP_RUN_ID**(*filename*)

HErmes.utils.files.**GCD**(*filename*)

HErmes.utils.files.**SIM_RUN_ID**(*filename*)

HErmes.utils.files.**check_hdf_integrity**(*infiles*, *checkfor=None*)
Checks if hdfiles can be openend and returns a tuple integer_files,corrupt_files

Parameters **infiles** (*list*) –

Keyword Arguments **checkfor** (*str*) –

HErmes.utils.files.**group_names_by_regex**(*names*, *regex=<function <lambda>>*, *firstpattern=<function <lambda>>*, *estimate_first=<function <lambda>>*)
Generate lists with files which all have the same name patterns, group by regex

Parameters **names** (*list*) – a list of file names

Keyword Arguments

- **regex** (*func*) – a regex to group by
- **firstpattern** (*func*) – the leading element of each list
- **estimate_first** (*func*) – if there are servaral elements which match firstpattern, estimate which is the first

Returns names grouped by reges with first pattern as leading element

Return type list

HErmes.utils.files.**harvest_files**(*path*, *ending='.bz2'*, *sanitizer=<function <lambda>>*, *use_ls=False*, *prefix='dcap://'*)
Get all the files with a specific ending from a certain path

Parameters **path** (*str*) – a path on the filesystem to look for files

Keyword Arguments

- **ending** (*str*) – glob for files with this ending
- **sanitizer** (*func*) – clean the file list with a filter
- **use_ls** (*bool*) – use unix ls to compile the filelist
- **prefix** (*str*) – apply this prefix to the file names

Returns All files in path which match ending and are filtered by sanitizer

Return type list

HErmes.utils.files.**strip_all_endings**(*filename*)

Split a filename at the first dot and declare everything which comes after it and consists of 3 or 4 characters (including the dot) as "ending"

Parameters **filename** (*str*) – a filename which shall be split

Returns file basename + ending

Return type list

HErmes.utils.itools module

Tools for managing iterables

HErmes.utils.itools.**flatten**(*iterable_of_iterables*)

Concat an iterable of iterables in a single iterable, chaining its elements

Parameters **iterable_of_iterables** (*iterable*) – Currently supported is dimensionality of 2

Returns np.ndarray

HErmes.utils.itools.**slicer**(*list_to_slice*, *slices*)

Generator Slice a list in individual slices

Parameters

- **list_to_slice** (*list*) – a list of items
- **slices** (*int*) – how many lists will be sliced of

Returns slices of the given list

Return type list

HErmes.utils.logger module

A logger with customizable loglevel at runtime.

class HErmes.utils.logger.**AbstractCustomLoggerType**

Bases: type

A modified logging.logger. Do not use directly, but as metaclass. Whenever the logger is called, HErmes.utils.logger.LOGLEVEL is queried to check for a change in the loglevel and a new logger instance is created accordingly. Python inspect is used to inspect the stack.

class HErmes.utils.logger.**Logger**
Bases: object

A custom logger with loglevel changeable at runtime. To change the loglevel, set the HErmes.utils.logger.LOGLEVEL variable: 10 = DEBUG, 20 = INFO, 30 = WARNING

HErmes.utils.logger.**alertstring**(*x*)

HErmes.utils.logger.**get_logger**(*loglevel*)
A logger from python logging on steroids.

Parameters **loglevel** (*int*) – 10 = DEBUG, 20 = INFO, 30 = WARNING

Returns logging.Logger

Module contents

Miscellaneous tools

1.1.2 Module contents

A package for filtering datasets as common in high energy physics. Read data from hdf or root files, classify the data in different categories and provide and easy interface to easy access to the variables stored in the files.

The HErmes modules provides the following submodules:

- *selection* : Start from a .json configuration file to create a full fledged dataset which acts as a container for in different categories.
- *utils* : Aggregator for files and logging.
- *fitting* : Fit models to variable distributions with iminuit.
- *plotting* : Data visualization.
- *icecube_goodies* : Weighting for icecube datasets.
- *analysis* : convenient functions for data analysis and working with distributions.

CHAPTER 2

Indices and tables

- modindex
- Glossary

Python Module Index

h

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

T

U

V

W

X

Y

Z