kafe Documentation

Release 3.1alpha1

Daniel Savoiu

July 26, 2013

CONTENTS

Contents:

CHAPTER

ONE

KAFE

1.1 kafe Package

1.1.1 `kafe` Package

A Python package for fitting and plotting for use in physics lab courses.

This Python package allows fitting of user-defined functions to data. A dataset is represented by a *Dataset* object which stores measurement data as *NumPy* arrays. The uncertainties of the data are also stored in the *Dataset* as an *error matrix*, allowing for both correlated and uncorrelated errors to be accurately represented.

The constructor of a *Dataset* object accepts several keyword arguments and can be used to construct a *Dataset* out of data which has been loaded into *Python* as *NumPy* arrays. Alternatively, a plain-text representation of a *Dataset* can be read from a file.

Also provided are helper functions which construct a *Dataset* object from a file containing column data (one measurement per row, column order can be specified).

1.1.2 `constants` Module

`kafe.constants.`**`F_SIGNIFICANCE = 2`**
Set significance for returning results and errors N = rounding error to N significant digits and value to the same order of magnitude as the error.

`kafe.constants.`**`G_PADDING_FACTOR_X = 1.2`**
factor by which to expand *x* data range

`kafe.constants.`**`G_PADDING_FACTOR_Y = 1.2`**
factor by which to expand *y* data range

`kafe.constants.`**`G_PLOT_POINTS = 200`**
number of plot points for plotting the function

`kafe.constants.`**`M_CONFIDENCE_LEVEL = 0.05`**
Confidence level for hypythesis test. A fit is rejected it χ^2_{prob} is smaller than this constant

`kafe.constants.`**`M_MAX_ITERATIONS = 6000`**
Maximum *Minuit* iterations until aborting the process

`kafe.constants.`**`M_MAX_X_FIT_ITERATIONS = 2`**
Number of maximal additional iterations for *x* fit (0 disregards *x* errors)

`kafe.constants.`**`M_TOLERANCE = 0.1`**
Minuit tolerance level

1.1.3 dataset Module

class kafe.dataset.**Dataset**(***kwargs*)

The *Dataset* object is a data structure for storing measurement and error data. In this implementation, the *Dataset* has the compulsory field *data*, which is used for storing the measurement data, and another field *cov_mats*, used for storing the covariance matrix for each axis.

There are several ways a *Dataset* can be constructed. The most straightforward way is to specify an input file containing a plain-text representation of the dataset:

```
>>> my_dataset = Dataset(input_file='/path/to/file')
```

or

```
>>> my_dataset = Dataset(input_file=my_file_object)
```

If an *input_file* keyword is provided, all other input is ignored. The *Dataset* plain-text representation format is as follows:

```
# x data
x_1  sigma_x_1
x_2  sigma_x_2  cor_x_12
...  ...        ...       ...
x_N  sigma_x_N  cor_x_1N  ...  cor_x_NN

# y data
y_1  sigma_y_1
y_2  sigma_y_2  cor_y_12
...  ...        ...       ...
y_N  sigma_y_N  cor_y_1N  ...  cor_y_NN
```

Here, the *sigma_...* represents the statistical error of the data point and *cor_..._ij* is the correlation coefficient between the *i*-th and *j*-th data point.

Alternatively, field data can be set by passing iterables as keyword arguments. Available keywords for this purpose are:

data

a tuple/list of measurement data. Each element of the tuple/list must be iterable and be of the same length. The first element of the **data** tuple/list is assumed to be the *x* data, and the second to be the *y* data:

```
>>> my_dataset = Dataset(data=([0., 1., 2., 3., 4.], [1.23, 3.45, 5.62, 7.88, 9.64]))
```

Alternatively, x-y value pairs can also be passed as **data**. The following is equivalent to the above:

```
>>> my_dataset = Dataset(data=([0.0, 1.23], [1.0, 3.45], [2.0, 5.62], [3.0, 7.88], [4.0, 9.6
```

In case the *Dataset* contains two data points, the ordering is ambiguous. In this case, the first ordering (*x* data first, then *y* data) is assumed.

cov_mats

a tuple/list of two-dimensional iterables containing the covariance matrices for *x* and *y*, in that order. Covariance matrices can be any sort of two-dimensional NxN iterables, assuming N is the number of data points.

```
>>> my_dataset = Dataset(data=([0., 1., 2.], [1.23, 3.45, 5.62]), cov_mats=(my_cov_mat_x, my
```

This keyword argument can be omitted, in which case covariance matrices of zero are assumed. To specify a covariance matrix for a single axis, replace the other with `None`.

```
>>> my_dataset = Dataset(data=([0., 1., 2.], [1.23, 3.45, 5.62]), cov_mats=(None, my_cov_mat
```

title

the name of the *Dataset*. If omitted, the *Dataset* will be given the generic name 'Untitled Dataset'.

axis_labels = None
axis labels

axis_units = None
units to assume for axis

cov_mat_is_regular(*axis*)
Returns *True* if the covariance matrix for an axis is regular and `False` if it is singular.

axis [string or int] Axis for which to check for regularity of the covariance matrix. Can be `'x'` or `'y'`.

cov_mats = None
list of covariance matrices

data = None
list containing measurement data (axis-ordering)

get_axis(*axis_alias*)
Get axis id from an alias.

axis_alias Alias of the axis whose id should be returned. This is for example either `'0'` or `'x'` for the *x*-axis (id 0).

get_cov_mat(*axis*, *fallback_on_singular=None*)
Get the error matrix for an axis.

axis string or int Axis for which to load the error matrix. Can be `'x'` or `'y'`. Type: string

fallback_on_singular [*numpy.matrix* or string] What to return if the matrix is singular. If this is `None` (default), the matrix is returned anyway. If this is a *numpy.matrix* object or similar, that is returned istead. Alternatively, the shortcuts `'identity'` or `1` and `'zero'` or `0` can be used to return the identity and zero matrix respectively.

get_data(*axis*)
Get the measurement data for an axis.

axis Axis for which to get the measurement data. Can be `'x'` or `'y'`. Type: string

get_data_span(*axis*, *include_error_bars=False*)
Get the data span for an axis. The data span is a tuple (*min*, *max*) containing the smallest and highest coordinates for an axis.

axis Axis for which to get the data span. Can be `'x'` or `'y'`. Type: string

include_error_bars [bool] `True` if the returned span should be enlarged to contain the error bars of the smallest and largest datapoints (default: `False`) Type: boolean

get_formatted(*format_string='.06e'*, *delimiter='t'*)
Returns the dataset in a plain-text format which is human-readable and can later be used as an input file for the creation of a new *Dataset*. The format is as follows:

```
# x data
x_1  sigma_x_1
x_2  sigma_x_2  cor_x_12
...  ...        ...       ...
```

```
x_N  sigma_x_N  cor_x_1N  ...  cor_x_NN

# y data
y_1  sigma_y_1
y_2  sigma_y_2  cor_y_12
...  ...        ...       ...
y_N  sigma_y_N  cor_y_1N  ...  cor_y_NN
```

Here, the `x_i` and `y_i` represent the measurement data, the `sigma_?_i` are the statistical uncertainties of each data point, and the `cor_?_ij` are the correlation coefficients between the *i*-th and *j*-th data point.

If the `x` or `y` errors are not correlated, then the entire correlation coefficient matrix can be omitted. If there are no statistical uncertainties for an axis, the second column can also be omitted. A blank line is required at the end of each data block!

format_string [string (optional)] A format string with which each entry will be rendered. Default is `'.06e'`, which means the numbers are represented in scientific notation with six significant digits.

delimiter [string (optional)] A delimiter used to separate columns in the output.

get_size()
Get the size of the *Dataset*. This is equivalent to the length of the *x*-axis data.

has_correlations(*axis*)
Returns *True* if the covariance matrix for an axis is regular and `False` if it is singular.

axis string or int Axis for which to check for regularity of the covariance matrix. Can be `'x'` or `'y'`.

has_errors(*axis*)
Returns *True* if the covariance matrix for an axis is regular and `False` if it is singular.

axis string or int Axis for which to check for regularity of the covariance matrix. Can be `'x'` or `'y'`.

n_axes = None
dimensionality of the *Dataset*. Currently, only 2D *Datasets* are supported

n_datapoints = None
number of data points in the *Dataset*

read_from_file(*input_file*)
Reads the *Dataset* object from a file.

returns [boolean] `True` if the read succeeded, `False` if not.

set_cov_mat(*axis*, *mat*)
Set the error matrix for an axis.

axis Axis for which to load the error matrix. Can be `'x'` or `'y'`.

mat Error matrix for the axis. Passing *None* unsets the error matrix. Type: *numpy.matrix* or *None*

set_data(*axis*, *data*)
Set the measurement data for an axis.

axis Axis for which to set the measurement data. Can be `'x'` or `'y'`. Type: string

data Measurement data for axis. Type: any iterable

write_formatted(*file_path*, *format_string='.06e'*, *delimiter='t'*)
Writes the dataset to a plain-text file. For details on the format, see get_formatted.

file_path [string] Path of the file object to write. **WARNING**: *overwrites existing files*!

format_string [string (optional)] A format string with which each entry will be rendered. Default is `'.06e'`, which means the numbers are represented in scientific notation with six significant digits.

delimiter [string (optional)] A delimiter used to separate columns in the output.

kafe.dataset.**build_dataset**(*xdata*, *ydata*, ***kwargs*)

This helper function creates a *Dataset* from a series of keyword arguments.

Valid keyword arguments are:

xdata and ydata These keyword arguments are mandatory and should be iterables containing the measurement data.

error specification keywords A valid keyword is composed of an axis (*x* or *y*), an error relativity specification (*abs* or *rel*) and error correlation type (*stat* or *syst*). The errors are then set as follows:

1. **For statistical errors:**
   - if keyword argument is a *NumPy* array, the error list is set to that
   - if keyword argument is a number, an error list with identical entries is generated
2. **For systematic errors:**
   - keyword argument *must* be a single number. The global correlated error for the acis is then set to that.

So, for example:

```
>>> myDataset = build_dataset(..., yabsstat=0.3, yrelsyst=0.1)
```

creates a dataset where the statistical error of each *y* coordinate is set to 0.3 and the overall systematic error of *y* is set to 0.1.

kafe.dataset.**debug_print**(*message*)

1.1.4 file_tools Module

kafe.file_tools.**parse_column_data**(*file_to_parse*, *field_order='x, y'*, *delimiter=' '*)

Parses a file which contains measurement data in a one-measurement-per-row format. The field (column) order can be specified. It defaults to *x,y'*. *Valid field names are 'x*, *y*, *xabsstat*, *yabsstat*, *xrelstat*, *yrelstat*, *xabssyst*. Another valid field name is *ignore* which can be used to skip a field.

Every valid measurement data file *must* have an *x* and a *y* field.

Additionally, a delimiter can be specified. If this is a whitespace character or omitted, any sequence of whitespace characters is assumed to separate the data.

file_to_parse [file-like object or string containing a file path] The file to parse.

field_order [string (optional)] A string of comma-separated field names giving the order of the columns in the file. Defaults to `'x,y'`.

delimiter [string (optional)] The field delimiter used in the file.

1.1.5 fit Module

class kafe.fit.**Fit**(*dataset*, *fit_function*, *external_fcn=<function chi2 at 0x3823de8>*, *function_label=None*, *function_equation=None*)

Object representing a fit. This object references the fitted *Dataset*, the fit function and the resulting fit parameters.

Necessary arguments are a *Dataset* object and a fit function (which should be fitted to the *Dataset*). Optionally, an external function *FCN* (whose minima should be located to find the best fit) can be specified. If not given, the *FCN* function defaults to χ^2.

dataset [*Dataset*] A *Dataset* object containing all information about the data

fit_function [function] A user-defined Python function to be fitted to the data. This function's first argument must be the independent variable *x*. All other arguments *must* be named and have default values given. These defaults are used as a starting point for the actual minimization. For example, a simple linear function would be defined like:

```
>>> def linear_2par(x, slope=1, y_intercept=0):
...     return slope * x + y_intercept
```

Be aware that choosing sensible initial values for the parameters is often crucial for a succesful fit, particularly for functions of many parameters.

external_fcn [function (optional)] An external *FCN* (function to minimize). This function must have the following call signature:

```
>>> FCN(xdata, ydata, cov_mat, fit_function, param_values)
```

It should return a float. If not specified, the default χ^2 *FCN* is used. This should be sufficient for most fits.

function_label [LaTeX-formatted string (optional)] A name/label/short description of the fit function. This appears in the legend describing the fitter curve. If omitted, this defaults to the function's Python name.

function_equation [LaTeX-formatted string (optional)] The fit function's equation.

call_external_fcn(**param_values*)
Wrapper for the external *FCN*. Since the actual fit process depends on finding the right parameter values and keeping everything else constant, we can use the *Dataset* object to pass known, fixed information to the external *FCN*, varying only the parameter values.

param_values [sequence of values] the parameter values at which *FCN* is to be evaluated

do_fit(*quiet=False*, *verbose=False*)
Runs the fit algorithm for this *Fit* object.

First, the *Dataset* is fitted considering only uncertainties in the *y* direction. If the *Dataset* has no uncertainties in the *y* direction, they are assumed to be equal to 1.0 for this preliminary fit, as there is no better information available.

Next, the fit errors in the *x* direction (if they exist) are taken into account by projecting the covariance matrix for the *x* errors onto the *y* covariance matrix. This is done by taking the first derivative of the fit function in each point and "projecting" the *x* error onto the resulting tangent to the curve.

This last step is repeater until the change in the error matrix caused by the projection becomes negligible.

quiet [boolean] Set to `True` if no output should be printed.

verbose [boolean] Set to `True` if more output should be printed.

fit_one_iteration(*verbose=False*)
Instructs the minimizer to do a minimization.

get_current_fit_function()
This method returns a function object corresponding to the fit function for the current parameter values. The returned function is a function of a single variable.

returns [function] A function of a single variable corresponding to the fit function at the current parameter values.

get_error_matrix()
This method returns the covariance matrix of the fit parameters which is obtained by querying the minimizer object for this fit

returns [*numpy.matrix*] The covariance matrix of the parameters.

get_parameter_errors(*rounding=False*)
Get the current parameter uncertainties from the minimizer.

rounding [boolean] Whether or not to round the returned values to significance.

returns [tuple] A tuple of the parameter uncertainties

get_parameter_values(*rounding=False*)
Get the current parameter values from the minimizer.

rounding [boolean] Whether or not to round the returned values to significance.

returns [tuple] A tuple of the parameter values

print_fit_details()
prints some fit goodness details

print_fit_results()
prints fit results

print_rounded_fit_parameters()
prints the fit parameters

project_x_covariance_matrix()
Project the *x* errors from the *x* covariance matrix onto the total matrix.

This is done elementwise, according to the formula:

$$C_{\text{tot},ij} = C_{y,ij} + C_{x,ij} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j}$$

kafe.fit.**chi2**(*xdata*, *ydata*, *cov_mat*, *fit_function*, *param_values*)
A simple χ^2 implementation. Calculates χ^2 according to the formula:

$$\chi^2 = \lambda^T C^{-1} \lambda$$

Here, λ is the residual vector $\lambda = \vec{y} - \vec{f}(\vec{x})$ and C is the covariance matrix.

xdata [iterable] The *x* measurement data

ydata [iterable] The *y* measurement data

cov_mat [*numpy.matrix*] The total covariance matrix

fit_function [function] The fit function $f(x)$

param_values [list/tuple] The values of the parameters at which $f(x)$ should be evaluated.

kafe.fit.**round_to_significance**(*value*, *error*, *significance=2*)
Rounds the error to the established number of significant digits, then rounds the value to the same order of magnitude as the error.

value [float] value to round to significance

error [float] uncertainty of the value

significance [int (optional)] number of significant digits of the error to consider

1.1.6 function_tools Module

kafe.function_tools.**derivative**(*func*, *derive_by_index*, *variables_tuple*, *derivative_spacing*)
Gives $\frac{\partial f}{\partial x_k}$ for $f = f(x_0, x_1, \ldots)$. *func* is f, *variables_tuple* is $\{x_i\}$ and *derive_by_index* is k.

kafe.function_tools.**derive_by_parameters**(*func*, *x_0*, *param_list*, *derivative_spacing*)
Returns the gradient of *func* with respect to its parameters, i.e. with respect to every variable of *func* except the first one.

kafe.function_tools.**derive_by_x**(*func*, *x_0*, *param_list*, *derivative_spacing*)
If *x_0* is iterable, gives the array of derivatives of a function $f(x, par_1, par_2, \ldots)$ around $x = x_i$ at every x_i in $\vec{x}$. If *x_0* is not iterable, gives the derivative of a function $f(x, par_1, par_2, \ldots)$ around $x =$ x_0.

kafe.function_tools.**get_function_property**(*func*, *prop*)
Returns a specific property of the function. This assumes that the function is defined as

```
>>> def func(x, par1=1.0, par2=3.14, par3=2.71, ...): ...
```

func [function] A function object from which to extract the property.

prop [any of 'name', 'parameter names', 'parameter defaults', 'number of parameters'] A string representing a property.

kafe.function_tools.**outer_product**(*input_array*)
Takes a *NumPy* array and returns the outer (dyadic, Kronecker) product with itself. If *input_array* is a vector $\mathbf{x}$, this returns $\mathbf{x}\mathbf{x}^T$.

1.1.7 minuit Module

class kafe.minuit.**Minuit**(*number_of_parameters*, *function_to_minimize*, *par_names*, *start_params*, *param_errors*, *quiet=True*, *verbose=False*)
Bases: object

A class for communicating with ROOT's function minimizer tool Minuit.

FCN_wrapper(*number_of_parameters*, *derivatives*, *f*, *parameters*, *internal_flag*)
This is actually a function called in ROOT and acting as a C wrapper for our B{FCN}, which is implemented in Python.

This function is called by Minuit several times during a fit. It doesn't return anything but sets modifies one of its arguments (*f*). This is ugly. Its argument structure is fixed and determined by Minuit:

@param number_of_parameters: The number of parameters of the current fit @type number_of_parameters: int

@param derivatives: computed gradient ?? @type derivatives: ??

@param f: A (C-compatible) array. The desired function value is in f[0] after execution. @type f: array

@param parameters: A C array of parameters. Is cast to a Python list @type parameters: array

@param internal_flag: A flag allowing for different behaviour of the function @type internal_flag: any int from M{1} (initial run) to M{4} (normal run)

get_chi2_probability(*n_deg_of_freedom*)
Returns the probability that an observed S{chi}-square exceeds the calculated value of S{chi}-square for this fit by chance, even for a correct model. In other words, returns the probability that a worse fit of the model to the data exists. If this is a small value (typically M{< 5%}), this means the fit is pretty bad. For values below this threshold, the model very probably does not fit the data.

get_error_matrix()
Retrieves the parameter error matrix from TMinuit

get_fit_info(*info*)
Retrieves other info from Minuit. The argument *info' can be any of the following: - 'fcn*: *FCN* value at

minimum, - *edm*: estimated distance to minimum - *err_def*: *Minuit* error matrix status code - *status_code*: *Minuit* general status code

get_parameter_errors()
Retrieves the parameter errors from TMinuit. Returns a tuple.

get_parameter_info()
Retrieves parameter information from TMinuit. Returns a list of tuples (param_name, param_val, param_error)

get_parameter_name(*param_nr*)
Sets the fit parameters. If param_values='None', tries to infer defaults from the function_to_minimize.

get_parameter_values()
Retrieves the parameter values from TMinuit. Returns a tuple.

minimize()
Do the minimization

reset()

set_err(*up_value=1.0*)
Sets the UP value for Minuit. Default: 1.0 (good for chi2)

set_parameter_errors(*param_errors=None*)
Sets the fit parameter errors. If param_values='None', sets the error to 1% of the parameter value.

set_parameter_names(*param_names*)
Sets the fit parameters. If param_values='None', tries to infer defaults from the function_to_minimize.

set_parameter_values(*param_values*)
Sets the fit parameters. If param_values='None', tries to infer defaults from the function_to_minimize.

set_print_level(*print_level=3*)
Sets the print level for Minuit. Default: 0 (suppress all output)

set_strategy(*strategy_id=1*)
Sets the strategy Minuit. Default: 1 (optimized)

1.1.8 numeric_tools Module

kafe.numeric_tools.**cor_to_cov**(*cor_mat*, *error_list*)
Converts a correlation matrix to a covariance matrix according to the formula

$$\mathrm{Cov}_{ij} = \mathrm{Cor}_{ij}\,\sigma_i\,\sigma_j$$

cor_mat [*numpy.matrix*] The correlation matrix to convert.

error_list [sequence of floats] A sequence of statistical errors. Must be of the same length as the diagonal of *cor_mat*.

kafe.numeric_tools.**cov_to_cor**(*cov_mat*)
Converts a covariance matrix to a correlation matrix according to the formula

$$\mathrm{Cor}_{ij} = \frac{\mathrm{Cov}_{ij}}{\sqrt{\mathrm{Cov}_{ii}\,\mathrm{Cov}_{jj}}}$$

cov_mat The covariance matrix to convert. Type: *numpy.matrix*

kafe.numeric_tools.**extract_statistical_errors**(*cov_mat*)
Extracts the statistical errors from a covariance matrix. This means it returns the (elementwise) square root of the diagonal entries

cov_mat The covariance matrix to extract errors from. Type: *numpy.matrix*

kafe.numeric_tools.**make_symmetric_lower**(*mat*)
Copies the matrix entries below the main diagonal to the upper triangle half of the matrix. Leaves the diagonal unchanged. Returns a *NumPy* matrix object.

mat [*numpy.matrix*] A lower diagonal matrix.

returns [*numpy.matrix*] The lower triangle matrix.

kafe.numeric_tools.**zero_pad_lower_triangle**(*triangle_list*)
Converts a list of lists into a lower triangle matrix. The list members should be lists of increasing length from 1 to N, N being the dimension of the resulting lower triangle matrix. Returns a *NumPy* matrix object.

For example:

```
>>> zero_pad_lower_triangle([ [1.0], [0.2, 1.0], [0.01, 0.4, 3.0] ])
matrix([[ 1.  ,  0.  ,  0.  ],
        [ 0.2 ,  1.  ,  0.  ],
        [ 0.01,  0.4 ,  3.  ]])
```

triangle_list [list] A list containing lists of increasing length.

returns [*numpy.matrix*] The lower triangle matrix.

1.1.9 `plot` Module

class kafe.plot.**Plot**(**fits*, ***kwargs*)

compute_plot_range(*include_error_bars=True*)
Compute the span of all child datasets and sets the plot range to that

extend_span(*axis*, *new_span*)
Expand the span of the current plot.

This method extends the current plot span to include *new_span*

init_plots()
Initialize the plots for each fit.

plot(*p_id*)
Plot the *Fit* object with the number *p_id* to its figure.

plot_all()
Plot every *Fit* object to its figure.

save(*output_file*)
Save the *Plot* to a file.

show()
Show the *Plot* in a matplotlib interactive window.

class kafe.plot.**PlotStyle**
Class for specifying a style for a specific plot. This object stores a progression of marker and line types and colors, as well as preferences relating to point size and label size. These can be overriden by overwriting the instance variables directly. A series of *get_...* methods are provided which go through these lists cyclically.

get_line(*idm*)
Get a specific line type. This runs cyclically through the defined defaults.

get_linecolor(*idm*)
Get a specific line color. This runs cyclically through the defined defaults.

get_marker(*idm*)
Get a specific marker type. This runs cyclically through the defined defaults.

get_markercolor(*idm*)
Get a specific marker color. This runs cyclically through the defined defaults.

get_pointsize(*idm*)
Get a specific point size. This runs cyclically through the defined defaults.

kafe.plot.**pad_span**(*span*, *pad_coeff=1*, *additional_pad=None*)
Enlarges the interval *span* (list of two floats) symmetrically around its center to length *pad_coeff*. Optionally, an *additional_pad* argument can be specified. The returned span is then additionally enlarged by that amount.

additional_pad can also be a list of two floats which specifies an asymmetric amount by which to enlarge the span. Note that in this case, positive entries in *additional_pad* will enlarge the span (move the interval end away from the interval's center) and negative amounts will shorten it (move the interval end towards the interval's center).

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX

M

N

O

P

R

S

W

Z