
PhyPraKit Documentation

Release 1.0

Günter Quast

May 21, 2017

CONTENTS

1 Übersicht:	1
2 Indices and tables	3
3 Darstellung und Auswertung von Messdaten	5
4 Dokumentation der Beispiele	7
5 Modul-Dokumentation	9
Python Module Index	23
Index	25

ÜBERSICHT:

PhyPraKit ist eine Sammlung nützlicher Funktionen in der Sprache *python* (vers. 2.7) zur Visualisierung und Auswertung von Daten in den physikalischen Praktika. Beispiele illustrieren jeweils die Anwendung.

Version der Dokumentation vom Mai 2017

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

DARSTELLUNG UND AUSWERTUNG VON MESSDATEN

In allen Praktika zur Physik werden Methoden zur Darstellung und Auswertung von Messdaten benötigt. Die Script- und Programmiersprache *python* mit den Zusatzpaketen *numpy* und *matplotlib* ist ein universelles Werkzeug, um die Wiederholbarkeit und Reproduzierbarkeit von Datenauswertungen zu gewährleisten.

In der Veranstaltung “Computergestützte Datenauswertung” (<http://www.ekp.kit.edu/~quast/CgDA>), die im neuen Studienplan für den Bachelorstudiengang Physik am KIT seit dem Sommersemester 2016 angeboten wird, werden Methoden und Software zur grafischen Darstellung von Daten, deren Modellierung und Auswertung eingeführt.

Die folgenden Links erlauben einen schnellen Überblick über die Inhalte der Vorlesung und den Beispielen aus den Übungen:

- **Zusammenfassung der Vorlesung und Dokumentation der Code-Beispiele** http://www.ekp.kit.edu/~quast/CgDA/CgDA-html/CgDA_ZusFas.html
- **Installation der Software auf verschiedenen Plattformen**
 - Dokumentation in html: <http://www.ekp.kit.edu/~quast/CgDA/CgDA-SoftwareInstallation-html>
 - Dokumentation in pdf: <http://www.ekp.kit.edu/~quast/CgDA/CgDA-SoftwareInstallation.pdf>
 - Softwarepakete: <http://www.ekp.kit.edu/~quast/CgDA/Software>

Speziell für das “Praktikum zur klassischen Physik” finden sich eine kurze Einführung (http://www.ekp.kit.edu/~quast/CgDA/PhysPrakt/CgDA_APraktikum.pdf) sowie die hier dokumentierten einfachen Beispiele als Startpunkt für eigene Auswertungen (<http://www.ekp.kit.edu/~quast/CgDA/PhysPrakt/>).

Die vorliegende Sammlung von Funktionen im Paket *PhyPraKit* enthält Funktionen zum Einlesen von Daten aus diversen Quellen, zur Datenvisualisierung, Signalbearbeitung und statistischen Datenauswertung und Modellanpassung sowie Werkzeuge zur Erzeugung simulierter Daten. Dabei wurde absichtlich Wert auf eine einfache, die Prinzipien unterstreichende Codierung gelegt und nicht der möglichst effizienten bzw. allgemeinsten Implementierung der Vorzug gegeben.

DOKUMENTATION DER BEISPIELE

PhyPraKit.py ist ein Paket mit nützlichen Hilfsfunktionen

zum import in eigene Beispielen mittels:

```
import PhyPraKit as ppk
```

oder:

```
from PhyPraKit import ...
```

- *test_readColumnData.py* ist ein Beispiel zum Einlesen von Spalten aus Textdateien; die zugehörigen *Meta-daten* können ebenfalls an das Script übergeben werden und stehen so bei der Auswertung zur Verfügung.
- *test_readtxt.py* liest Ausgabedateien im allgemeinem .txt-Format
 - Entfernen aller ASCII-Sonderzeichen außer dem Spalten-Trenner
 - Ersetzen des deutschen Dezimalkommas durch Dezimalpunkt
- *test_readPicoScope.py* liest Ausgabedateien von PicoScpe im .txt-Format
- *test_labxParser.py* liest Ausgabedateien von Leybold CASSY im .labx-Format. Die Kopfzeilen und Daten von Messreihen werden als Listen in python zur Verfügung gestellt.
- *test_AutoCorrelation.py* liest die Datei *AudioData.csv* und führt ein Analyse der Autokorrelation zur Frequenzbestimmung durch.
- *test_kRegression.py* dient zur Anpassung einer Geraden an Messdaten mit Fehlern in Ordinaten- und Abszissenrichtung und mit allen Messpunkten gemeinsamen (d. h. korrelierten) relativen oder absoluten systematischen Fehlern mit dem Paket *kafe*.
- *test_linRegression.py* ist eine einfachere Version mit *python*-Bordmitteln zur Anpassung einer Geraden an Messdaten mit Fehlern in Ordinaten- und Abszissenrichtung. Korrelierte Unsicherheiten werden nicht unterstützt.
- *test_kFit.py* ist eine verallgemeinerte Version von *test_kRegression* und dient zur Anpassung einer beliebigen Funktion an Messdaten mit Fehlern in Ordinaten- und Abszissenrichtung und mit allen Messpunkten gemeinsamen (d. h. korrelierten) relativen oder absoluten systematischen Fehlern mit dem Paket *kafe*.
- *test_Histogram.py* ist ein Beispiel zur Darstellung und statistischen Auswertung von Häufigkeitsverteilungen (Histogrammen) in einer und zwei Dimensionen.
- *test_generateXYata.py* zeigt, wie man mit Hilfe von Zufallszahlen “künstliche Daten” zur Veranschaulichung oder zum Test von Methoden zur Datenauswertung erzeugen kann.

Weitere python-Skripte als Beisepiele zur Anwendung von Modulen in *PhyPraKit*:

- *kfitf.py* ist ein Kommandozeilen-Werkzeug, mit dem man komfortabel Anpassungen ausführen kann, bei denen Daten und Fit-Funktion in einer einzigen Datei angegeben werden. Beispiele finden sich in den Dateien mit der Endung *.fit*.
- *Beispiel_Drehpendel.py* demonstriert die Analyse von am Drehpendel mit CASSY aufgenommenen Daten. Enthalten sind einfache Funktionen zum Filtern und bearbeiten der Daten, Suche nach Extrema, zur Anpassung einer Einhüllenden, zur diskreten Fourier-Transformation und zur Interpolation von Messdaten mit kubischen Spline-Funktionen.
- *Beispiel_Hysteresep.py* demonstriert die Analyse von Daten, die mit einem USB-Oszilloskop der Marke *PicoScope* am Versuch zur Hysteresis aufgenommen wurden. Die aufgezeichneten Werte für Strom und B-Feld werden in einen Zweig für steigenden und fallenden Strom aufgeteilt, mit Hilfe von kubischen Splines interpoliert und dann integriert.
- *Beispiel_Wellenform.py* zeigt eine typische Auswertung periodischer Daten am Beispiel der akustischen Anregung eines Metallstabs. Genutzt werden Fourier-Transformation und eine Suche nach charakteristischen Extrema. Die Zeitdifferenzen zwischen deren Auftreten im Muster werden bestimmt, als Häufigkeitsverteilung dargestellt und die Verteilungen statistisch ausgewertet.
- *Beispiel_GammaSpektroskopie.py* liest mit dem Vielkanalanalysator des CASSY-Systems aufgezeichnete Spektren aus einer im *.labx*-Format gespeicherten Datei ein.

Die übrigen *python*-Scripte im Verzeichnis wurden zur Erstellung der in der einführenden Vorlesung gezeigten Grafiken verwendet.

Für die **Erstellung von Protokollen** mit Tabellen, Grafiken und Formeln bietet sich das Textsatz-System *LaTeX* an. Die Datei *Protokollvorlage.zip* enthält eine sehr einfach gehaltene Vorlage, die für eigene Protokolle verwendet werden kann. Eine sehr viel umfangreichere Einführung sowie ein ausführliches Beispiel bietet die Fachschaft Physik unter dem Link <https://fachschaft.physik.kit.edu/drupal/content/latex-vorlagen>

MODUL-DOKUMENTATION

PhyPraKit a collection of tools for data handling, visualisation and analysis in Physics Lab Courses, recommended for “Physikalisches Praktikum am KIT”

PhyPraKit for Data Handling, Visualisation and Analysis

contains the following functions:

1. Data input:
 - `readColumnData()` read data and meta-data from text file
 - `readCSV()` read data in csv-format from file with header
 - `readtxt()` read data in “txt”-format from file with header
 - `readPicoScope()` read data from PicoScope
 - `readCassy()` read CASSY output file in .txt format
 - `labxParser()` read CASSY output file, .labx format
 - `writeCSV()` write data in csv-format (opt. with header)
2. signal processing:
 - `offsetFilter()` subtract an offset in array a
 - `meanFilter()` apply sliding average to smoothen data
 - `resample()` average over n samples
 - `Fourier_fft()` fast Fourier transformation of an array
 - `FourierSpectrum()` Fourier transformation of an array (slow, preferably use `fft` version)
 - **`simplePeakfinder()` find peaks and dips in an array comparing** neighbouring samples use `convolutionPeakfinder`
 - `convolutionPeakfinder()` find peaks and dips in an array
 - `atocorrelate()` autocorrelation function
3. statistics:
 - `wmean()` weighted mean
4. histograms tools:
 - `barstat()` statistical information bar chart
 - `nhist()` histogram plot based on `np.histogram()` and `plt.bar()` use `matplotlib.pyplot.hist()` instead

- `histstat()` statistical information from 1d-histogram
- `nhist2d()` 2d-histogram plot based on `np.histogram2d`, `plt.colormesh()` use `matplotlib.pyplot.hist2d()` instead
- `hist2dstat()` statistical information from 1d-histogram
- `profile2d()` “profile plot” for 2d data
- `chi2p_indep2d()` chi2 test on independence of data

5. linear regression:

- `linRegression()` linear regression, $y=ax+b$, with analytical formula
- `linRegressionXY()` linear regression, $y=ax+b$, with x and y errors ! deprecated, use ``odFit`` with linear model instead
- `odFit()` fit function with x and y errors (scipy ODR)
- `kRegression()` regression, $y=ax+b$, with x -, y - and correlated errors ! deprecated, use ``kFit`` with linear model instead
- `kFit()` fit function with x -, y - and correlated errors (kafe)

6. simulated data with MC-method:

- `smearData()` add random deviations to input data
- `generateXYdata()` generate simulated data

`PhyPraKit.PhyPraKit.FourierSpectrum(t, a, fmax=None)`

Fourier transform of amplitude spectrum $a(t)$, for equidistant sampling times (a simple implementation for didactical purpose only, consider using `Fourier_fft()`)

Args:

- t : np-array of time values
- a : np-array amplitude $a(t)$

Returns:

- arrays `freq`, `amp`: frequencies and amplitudes

`PhyPraKit.PhyPraKit.Fourier_fft(t, a)`

Fourier transform of the amplitude spectrum $a(t)$

method: uses `numpy.fft` and `numpy.fftfreq`; output amplitude is normalised to number of samples;

Args:

- t : np-array of time values
- a : np-array amplitude $a(t)$

Returns:

- arrays `f`, `a_f`: frequencies and amplitudes

`PhyPraKit.PhyPraKit.autocorrelate(a)`

calculate autocorrelation function of array

method: for array of length l , calculate $a[0]=\sum_{i=0}^{l-1} a[i]*a[i]$ $a[i]=1/a[0] * \sum_{k=0}^{l-i} a[i] * a[i+k-1]$ for $i=1, l-1$ uses inner product from numpy

Args:

- a: np-array

Returns

- np-array of len(a)

PhyPraKit.PhyPraKit.**barstat** (*bincont, bincent, pr=True*)
 statistics from a bar chart (histogram) with given bin contents and bin centres

Args:

- bincont: array with bin content
- bincent: array with bin centres

Returns:

- float: mean, sigma and sigma on mean

PhyPraKit.PhyPraKit.**chi2p_indep2d** (*H2d, bcx, bcy, pr=True*)
 perform a chi2-test on independence of x and y

Args:

- H2d: histogram array (as returned by histogram2d)
- bcx: bin contents x
- bcy: bin contents y

Returns:

- float: p-value w.r.t. assumption of independence

PhyPraKit.PhyPraKit.**convolutionPeakfinder** (*a, width=10, th=0.1*)

find positions of all Peaks and Dips in data (simple version for didactical purpose, consider using `scipy.signal.find_peaks_cwt()`)

method: convolute array a with signal template of given width and return extrema of convoluted signal, i.e. places where template matches best

Args:

- a: array-like, input data
- width: int, width of signal to search for
- th: float, relative threshold for peaks above minimum

Returns:

- pidx: list, indices (in original array) of peaks

PhyPraKit.PhyPraKit.**generateXYdata** (*xdata, model, sx, sy, mpar=None, srelx=None, srelx=None, xabscor=None, yabscor=None, xrelcor=None, yrelcor=None*)

Generate measurement data according to some model assumes xdata is measured within the given uncertainties; the model function is evaluated at the assumed “true” values xtrue, and a sample of simulated measurements is obtained by adding random deviations according to the uncertainties given as arguments.

Args:

- xdata: np-array, x-data (independent data)
- model: function that returns (true) model data (y-dat) for input x
- mpar: list of parameters for model (if any)

the following are single floats or arrays of length of x

- sx: gaussian uncertainty(ies) on x
- sy: gaussian uncertainty(ies) on y
- srelx: relative gaussian uncertainty(ies) on x
- srely: relative gaussian uncertainty(ies) on y

the following are common (correlated) systematic uncertainties

- xabscor: absolute, correlated error on x
- yabscor: absolute, correlated error on y
- xrelcor: relative, correlated error on x
- yrelcor: relative, correlated error on y

Returns:

- np-arrays of floats:
 - xtrue: true x-values
 - ytrue: true value = model(xtrue)
 - ydata: simulated data

`PhyPraKit.PhyPraKit.hist2dstat (H2d, xed, yed, pr=True)`
calculate statistical information from 2d Histogram

Args:

- H2d: histogram array (as returned by histogram2d)
- xed: bin edges in x
- yed: bin edges in y

Returns:

- float: mean x
- float: mean y
- float: variance x
- float: variance y
- float: covariance of x and y
- float: correlation of x and y

`PhyPraKit.PhyPraKit.histstat (binc, bine, pr=True)`
calculate mean of a histogram with bincontents binc and bin edges bine

Args:

- binc: array with bin content
- bine: array with bin edges

Returns:

- float: mean, sigma and sigma on mean

`PhyPraKit.PhyPraKit.kFit (func, x, y, sx, sy, p0=None, p0e=None, xabscor=None, yabscor=None, xrelcor=None, yrelcor=None, title='Daten', axis_labels=['X', 'Y'], plot=True, quiet=False)`

fit function `func` with errors on `x` and `y`; uses package *kafe*

Args:

- `func`: function to fit
- `x`: np-array, independent data
- `y`: np-array, dependent data

the following are single floats or arrays of length of `x`

- `sx`: uncertainty(ies) on `x`
- `sy`: uncertainty(ies) on `y`
- `p0`: array-like, initial guess of parameters
- `p0e`: array-like, initial guess of parameter uncertainties
- `xabscor`: absolute, correlated error(s) on `x`
- `yabscor`: absolute, correlated error(s) on `y`
- `xrelcor`: relative, correlated error(s) on `x`
- `yrelcor`: relative, correlated error(s) on `y`
- `title`: string, title of gaph
- `axis_labels`: List of strings, axis labels `x` and `y`
- `plot`: flag to switch off graphical ouput
- `quiet`: flag to suppress text and log output

Returns:

- np-array of float: parameter values
- np-array of float: parameter errors
- np-array: cor correlation matrix
- float: chi2 chi-square

`PhyPraKit.PhyPraKit.kRegression(x, y, sx, sy, xabscor=None, yabscor=None, xrelcor=None, yrelcor=None, title='Daten', axis_labels=['X', 'Y'], plot=True, quiet=False)`

linear regression $y(x) = ax + b$ with errors on `x` and `y`; uses package *kafe*

Args:

- `x`: np-array, independent data
- `y`: np-array, dependent data

the following are single floats or arrays of length of `x`

- `sx`: uncertainty(ies) on `x`
- `sy`: uncertainty(ies) on `y`
- `xabscor`: absolute, correlated error(s) on `x`
- `yabscor`: absolute, correlated error(s) on `y`
- `xrelcor`: relative, correlated error(s) on `x`

- yrelcor: relative, correlated error(s) on y
- title: string, title of gaph
- axis_labels: List of strings, axis labels x and y
- plot: flag to switch off graphical ouput
- quiet: flag to suppress text and log output

Returns:

- float: a slope
- float: b constant
- float: sa sigma on slope
- float: sb sigma on constant
- float: cor correlation
- float: chi2 chi-square

PhyPraKit.PhyPraKit.**labxParser** (*file*, *prlevel=1*)
read files in xml-format produced with Leybold CASSY

Args:

- file: input data in .labx format
- prlevel: control printout level, 0=no printout

Returns:

- list of strings: tags of measurmement vectors
- 2d list: measurement vectors read from file

PhyPraKit.PhyPraKit.**linRegression** (*x*, *y*, *sy*)
linear regression $y(x) = ax + b$

method: analytical formula

Args:

- x: np-array, independent data
- y: np-array, dependent data
- sx: np-array, uncertainty on y

Returns:

- float: a slope
- float: b constant
- float: sa sigma on slope
- float: sb sigma on constant
- float: cor correlation
- float: chi2 chi-square

PhyPraKit.PhyPraKit.**linRegressionXY** (*x*, *y*, *sx*, *sy*)
linear regression $y(x) = ax + b$ with errors on x and y uses numerical “orthogonal distance regression” from package scipy.odr

Args:

- x: np-array, independent data
- y: np-array, dependent data
- sx: np-array, uncertainty on y
- sy: np-array, uncertainty on y

Returns:

- float: a slope
- float: b constant
- float: sa sigma on slope
- float: sb sigma on constant
- float: cor correlation
- float: chi2 chi-square

`PhyPraKit.PhyPraKit.meanFilter(a, width=5)`

apply a sliding average to smoothen data,

method: value at index i and $\text{int}(\text{width}/2)$ neighbours are averaged to from the new value at index i

Args:

- a: np-array of values
- width: int, number of points to average over (if width is an even number, width+1 is used)

Returns:

- av smoothed signal curve

`PhyPraKit.PhyPraKit.nhist(data, bins=50, xlabel='x', ylabel='frequency')`

Histogram.hist show a one-dimensional histogram

Args:

- data: array containing float values to be histogrammed
- bins: number of bins
- xlabel: label for x-axis
- ylabel: label for y axis

Returns:

- float arrays bin content and bin edges

`PhyPraKit.PhyPraKit.nhist2d(x, y, bins=10, xlabel='x axis', ylabel='y axis', clabel='counts')`

Histogram.hist2d create and plot a 2-dimensional histogram

Args:

- x: array containing x values to be histogrammed
- y: array containing y values to be histogrammed
- bins: number of bins
- xlabel: label for x-axis
- ylabel: label for y axis

- clabel: label for colour index

Returns:

- float array: array with counts per bin
- float array: histogram edges in x
- float array: histogram edges in y

`PhyPraKit.PhyPraKit.odFit (fitf, x, y, sx, sy, p0=None)`

fit an arbitrary function with errors on x and y uses numerical “orthogonal distance regression” from package `scipy.odr`

Args:

- fitf: function to fit, arguments (array:P, float:x)
- x: np-array, independent data
- y: np-array, dependent data
- sx: np-array, uncertainty on x
- sy: np-array, uncertainty on y
- p0: none, scalar or array, initial guess of parameters

Returns:

- np-array of float: parameter values
- np-array of float: parameter errors
- np-array: cor correlation matrix
- float: chi2 chi-square

`PhyPraKit.PhyPraKit.offsetFilter (a)`

correct an offset in array a (assuming a symmetric signal around zero) by subtracting the mean

`PhyPraKit.PhyPraKit.profile2d (H2d, xed, yed)`

generate a profile plot from 2d histogram:

- mean y at a centre of x-bins, standard deviations as error bars

Args:

- H2d: histogram array (as returned by `histogram2d`)
- xed: bin edges in x
- yed: bin edges in y

Returns:

- float: array of bin centres in x
- float: array mean
- float: array rms
- float: array sigma on mean

`PhyPraKit.PhyPraKit.readCSV (file, nlhead=1)`

read Data in .csv format, skip header lines

Args:

- file: string, file name
- nhead: number of header lines to skip
- delim: column separator

Returns:

- hlines: list of string, header lines
- data: 2d array, 1st index for columns

PhyPraKit.PhyPraKit.**readCassy** (*file*, *prlevel=0*)
read Data exported from Cassy in .txt format

Args:

- file: string, file name
- prlevel: printout level, 0 means silent

Returns:

- units: list of strings, channel units
- data: tuple of arrays, channel data

PhyPraKit.PhyPraKit.**readColumnData** (*fname*, *cchar='#*', *delimiter=None*, *pr=True*)

read column-data from file

- input is assumed to be columns of floats
- characters following <cchar>, and <cchar> itself, are ignored
- words with preceeding '*' are taken as keywords for meta-data, text following the keyword is returned in a dictionary

Args:

- string fnam: file name
- int ncols: number of columns
- char delimiter: character separating columns
- bool pr: print input to std out if True

PhyPraKit.PhyPraKit.**readPicoScope** (*file*, *prlevel=0*)
read Data exported from PicoScope in .txt or .csv format

Args:

- file: string, file name
- prlevel: printout level, 0 means silent

Returns:

- units: list of strings, channel units
- data: tuple of arrays, channel data

PhyPraKit.PhyPraKit.**readtxt** (*file*, *nlhead=1*, *delim='\n'*)

read floating point data in general txt format skip header lines, replace decimal comma, remove special characters

Args:

- file: string, file name
- nhead: number of header lines to skip
- delim: column separator

Returns:

- hlines: list of string, header lines
- data: 2d array, 1st index for columns

`PhyPraKit.PhyPraKit.resample(a, t=None, n=11)`

perform average over n data points of array a, return reduced array, eventually with corresponding time values

method: value at index *i* and $\text{int}(\text{width}/2)$ neighbours are averaged to form the new value at index *i*

Args:

- a, t: np-arrays of values of same length
- width: int, number of values of array *a* to average over (if width is an even number, width+1 is used)

Returns:

- av: array with reduced number of samples
- tav: a second, related array with reduced number of samples

`PhyPraKit.PhyPraKit.simplePeakfinder(x, a, th=0.0)`

find positions of all Peaks and Dips in data x-coordinates are determined from weighted average over 3 data points

this only works for very smooth data with well defined extrema use `convolutionPeakfinder` instead

Args:

- x: np-array of values
- a: np-array of values at position x
- th: float, threshold for peaks

Returns:

- np-array: x positions of peaks as weighted mean over neighbours
- np-array: y positions

`PhyPraKit.PhyPraKit.smearData(d, s, srel=None, abscor=None, relcor=None)`

Generate measurement data from “true” input d by adding random deviations according to the uncertainties

Args:

- d: np-array, (true) input data

the following are single floats or arrays of length of array d

- s: gaussian uncertainty(ies) (absolute)
- srel: gaussian uncertainties (relative)

the following are common (correlated) systematic uncertainties

- absco: absolute, correlated uncertainty
- relco: relative, correlated uncertainty

Returns:

- np-array of floats: dm, smeared (=measured) data

`PhyPraKit.PhyPraKit.wmean(x, sx, pr=True)`
weighted mean of np-array x with uncertainties sx

Args:

- x: np-array of values
- sx: uncertainties
- pr: if True, print result

Returns:

- float: mean, sigma

`PhyPraKit.PhyPraKit.writeCSV(file, ldata, hlines=[], fmt='%10g')`
write data in .csv format, including header lines

Args:

- file: string, file name
- ldata: list of columns to be written
- hlines: list with header lines (optional)
- fmt: format string (optional)

Returns:

- 0/1 for success/fail

`test_readColumnData.py`

test data input from text file with module `PhyPraKit.readColumnData`

`test_readtxt.py`

uses `readtxt()` to read floating-point column-data in general .txt format

example output from PicoTech 8 channel data logger, with ‘ ‘ separated values, 2 header lines, german decimal comma and special character ‘^@’

`test_readPicoScope.py`

Einlesen von Daten aus einer mit PicoScope erstellten Datei

`cassylabxParser.py`

read files in xml-format produced with the Leybold Cassy system

uses `PhyPraKit.labxParser()`

`test_Histogram.py`

test histogram functionality in `PhyPraKit`

`test_AutoCorrelation.py`

test function *autocorrelate()* in PhyPraKit; determines the frequency of a periodic signal from maxima and minima of the autocorrelation function and performs statistical analysis of time between peaks/dips

uses *readCSV()*, *autocorrelate()*, *convolutionPeakfinder()* and *histstat()* from PhyPraKit

test_kRegression test linear regression with kafe using kFit from PhyPraKit uncertainties in x and y and correlated absolute and relative uncertainties

test_kFit test fitting an arbitrary function with kafe, with uncertainties in x and y and correlated absolute and relative uncertainties

test_generateDate

test generation of simulated data this simulates a measurement with given x-values with uncertainties; random deviations are then added to arrive at the true values, from which the true y-values are then calculated according to a model function. In the last step, these true y-values are smeared by adding random deviations to obtain a sample of measured values

kfitf.py

Perform a fit with the kafe package driven by input file

usage: kfitf.py [-h] [-n] [-s] [-c] [--noinfo] [-f FORMAT] filename

positional arguments: filename name of fit input file

optional arguments:

-h, --help	show this help message and exit
-n, --noplot	suppress output of plots on screen
-s, --saveplot	save plot(s) in file(s)
-c, --contour	plot contours and profiles
--noinfo	suppress fit info on plot
--noband	suppress 1-sigma band around function
--format FMT	graphics output format, default FMT = pdf

Beispiel_Drehpendel.py

Auswertung der Daten aus einer im CASSY labx-Format gespeicherten Datei am Beispiel des Drehpendels

- Einlesen der Daten im .labx-Format
- Säubern der Daten durch verschiedene Filterfunktionen: - offset-Korrektur - Glättung durch gleitenden Mittelwert - Zusammenfassung benachbarter Daten durch Mittelung
- Fourier-Transformation (einfach und fft)
- Suche nach Extrema (*peaks* und *dips*)
- Anpassung von Funktionen an Einhüllende der Maxima und Minima
- Interpolation durch Spline-Funktionen
- numerische Ableitung und Ableitung der Splines
- Phasenraum-Darstellung (aufgezeichnete Wellenfunktion gegen deren Ableitung nach der Zeit)

Beispiel_Hysteresis.py

Auswertung der Daten aus einer mit PicoScope erstellten Datei im txt-Format am Beispiel des Hysteresisversuchs

- Einlesen der Daten aus PicoScope-Datei vom Typ .txt oder .csv
- Darstellung Kanal_a vs. Kanal_b
- Auftrennung in zwei Zweige für steigenden bzw. abnehmenden Strom
- Interpolation durch kubische Splines
- Integration der Spline-Funktionen

test_readPicoScope.py

Einlesen von Daten aus dem mit PicoScope erstellten Dateien am Beispiel der akustischen Anregung eines Stabes

Beispiel_Drehpendel.py

Darstellung der Daten aus einer im CASSY labx-Format gespeicherten Datei am Beispiel der Gamma-Spektroskopie

- Einlesen der Daten im .labx-Format

PYTHON MODULE INDEX

b

Beispiel_Drehpendel, [20](#)
Beispiel_GammaSpektroskopie, [21](#)
Beispiel_Hysteresse, [20](#)
Beispiel_Wellenform, [21](#)

k

kfitf, [20](#)

p

PhyPraKit, [9](#)
PhyPraKit.PhyPraKit, [9](#)

t

test_AutoCorrelation, [19](#)
test_generateData, [20](#)
test_Histogram, [19](#)
test_kFit, [20](#)
test_kRegression, [20](#)
test_labxParser, [19](#)
test_readColumnData, [19](#)
test_readPicoScope, [19](#)
test_readtxt, [19](#)

A

autocorrelate() (in module PhyPraKit.PhyPraKit), 10

B

barstat() (in module PhyPraKit.PhyPraKit), 11
 Beispiel_Drehpendel (module), 20
 Beispiel_GammaSpektroskopie (module), 21
 Beispiel_Hysteresis (module), 20
 Beispiel_Wellenform (module), 21

C

chi2p_indep2d() (in module PhyPraKit.PhyPraKit), 11
 convolutionPeakfinder() (in module
 PhyPraKit.PhyPraKit), 11

F

Fourier_fft() (in module PhyPraKit.PhyPraKit), 10
 FourierSpectrum() (in module PhyPraKit.PhyPraKit), 10

G

generateXYdata() (in module PhyPraKit.PhyPraKit), 11

H

hist2dstat() (in module PhyPraKit.PhyPraKit), 12
 histstat() (in module PhyPraKit.PhyPraKit), 12

K

kFit() (in module PhyPraKit.PhyPraKit), 12
 kfitf (module), 20
 kRegression() (in module PhyPraKit.PhyPraKit), 13

L

labxParser() (in module PhyPraKit.PhyPraKit), 14
 linRegression() (in module PhyPraKit.PhyPraKit), 14
 linRegressionXY() (in module PhyPraKit.PhyPraKit), 14

M

meanFilter() (in module PhyPraKit.PhyPraKit), 15

N

nhist() (in module PhyPraKit.PhyPraKit), 15

nhist2d() (in module PhyPraKit.PhyPraKit), 15

O

odFit() (in module PhyPraKit.PhyPraKit), 16
 offsetFilter() (in module PhyPraKit.PhyPraKit), 16

P

PhyPraKit (module), 9
 PhyPraKit.PhyPraKit (module), 9
 profile2d() (in module PhyPraKit.PhyPraKit), 16

R

readCassy() (in module PhyPraKit.PhyPraKit), 17
 readColumnData() (in module PhyPraKit.PhyPraKit), 17
 readCSV() (in module PhyPraKit.PhyPraKit), 16
 readPicoScope() (in module PhyPraKit.PhyPraKit), 17
 readtxt() (in module PhyPraKit.PhyPraKit), 17
 resample() (in module PhyPraKit.PhyPraKit), 18

S

simplePeakfinder() (in module PhyPraKit.PhyPraKit), 18
 smearData() (in module PhyPraKit.PhyPraKit), 18

T

test_AutoCorrelation (module), 19
 test_generateData (module), 20
 test_Histogram (module), 19
 test_kFit (module), 20
 test_kRegression (module), 20
 test_labxParser (module), 19
 test_readColumnData (module), 19
 test_readPicoScope (module), 19
 test_readtxt (module), 19

W

wmean() (in module PhyPraKit.PhyPraKit), 19
 writeCSV() (in module PhyPraKit.PhyPraKit), 19