

---

# **PhyPraKit Documentation**

***Release 1.1.2***

**Günter Quast**

**Jan 05, 2021**



# CONTENTS

<b>1</b>	<b>About</b>	<b>3</b>
1.1	Installation: . . . . .	3
1.2	Übersicht: . . . . .	3
<b>2</b>	<b>Indices and tables</b>	<b>5</b>
<b>3</b>	<b>Darstellung und Auswertung von Messdaten</b>	<b>7</b>
<b>4</b>	<b>Dokumentation der Beispiele</b>	<b>9</b>
<b>5</b>	<b>Module Documentation</b>	<b>13</b>
	<b>Python Module Index</b>	<b>29</b>
	<b>Index</b>	<b>31</b>



*Version* 2021-01-05



## ABOUT

**PhyPraKit** is a collection of python modules for data visualisation and analysis in experimental laboratory courses in physics, in use at the faculty of physics at Karlsruhe Institute of Technology (KIT). As the modules are intended primarily for use by undergraduate students in Germany, the documentation is partly in German language, in particular the description of the examples.

Created by:

- Guenter Quast <guenter (dot) quast (at) online (dot) de>

A pdf version of this documentation is available here: [PhyPraKit.pdf](#).

## 1.1 Installation:

To use PhyPraKit, it is sufficient to place the file *PhyPraKit.py* in the same directory as the python scripts importing it.

Installation via *pip* is also supported. The recommendation is to use the installation package in the subdirectory *dist* and install in user space:

```
pip install --user --no-cache PhyPraKit<vers.>
```

## 1.2 Übersicht:

**PhyPraKit** ist eine Sammlung nützlicher Funktionen in der Sprache *python* (*vers. 2.7 oder  $\geq 3.4$* ) zum Aufnehmen, zur Bearbeitung, Visualisierung und Auswertung von Daten in den physikalischen Praktika. Die Anwendung der verschiedenen Funktionen des Pakets werden jeweils durch Beispiele illustriert.





## INDICES AND TABLES

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



## DARSTELLUNG UND AUSWERTUNG VON MESSDATEN

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

In der Veranstaltung “Computergestützte Datenauswertung” (<http://www.ekp.kit.edu/~quast/CgDA>), die im 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 die Beispielen aus den Übungen:

- **Zusammenfassung der Vorlesung und Dokumentation der Code-Beispiele** [http://www.ekp.kit.edu/~quast/CgDA/CgDA-html/CgDA\\_ZusFas.html](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](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 zur statistischen Datenauswertung und Modell Anpassung 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 Beispiele mittels:

```
import PhyPraKit as ppk
```

oder:

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

`PhyPraKit.py` enthält folgende Funktionen:

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)
- writeTexTable() write data in LaTeX table format

2. signal processing:

- offsetFilter() subtract an offset in an input array
- meanFilter() apply sliding average to smoothen data
- resample() average over n samples
- simplePeakfinder() find peaks and dips in an array recommend to use *convolutionPeakfinder*
- convolutionPeakfinder() find maxima (peaks) in an array
- convolutionEdgefinder() find maxima of slope (rising) edges in an array
- Fourier\_fft() fast Fourier transformation of an array
- FourierSpectrum() Fourier transformation of an array (*slow, preferably use fft version*)
- autocorrelate() autocorrelation function

3. statistics:

- wmean() weighted mean
- BuildCovarianceMatrix() build covariance matrix

- Cov2Cor() covariance matrix to correlation matrix
  - Cor2Cov() correlations + errors to covariance matrix
  - chi2prob() calculate chi<sup>2</sup> probability
4. histograms tools:
- barstat() statistical information (mean, sigma, error on mean) from 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
  - **kRegression() linear regression,  $y=ax+b$ , with (correlated) errors on x and y** ! deprecated, use ``kFit`` or ``k2Fit`` with linear model instead
  - odFit() fit function with x and y errors (scipy ODR)
  - mFit() fit with iminuit with correlated y errors, profile likelihood and contour lines
  - kFit() fit function with (correlated) errors on x and y (kafé)
  - k2Fit() fit function with (correlated) errors on x and y (kafé2)
6. simulated data with MC-method:
- smearData() add random deviations to input data
  - generateXYdata() generate simulated data

Die folgenden **Beispiele** illustrieren die Anwendung:

- *test\_readColumnData.py* ist ein Beispiel zum Einlesen von Spalten aus Textdateien; die zugehörigen *Metadaten* 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 USB-Oszilloskopen der Marke PicoScope im Format .csv oder .txt.
- *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\_convolutionFilter.py* liest die Datei *Wellenform.csv* und bestimmt Maxima und fallende Flanken des Signals

- *test\_AutoCorrelation.py* liest die Datei *AudioData.csv* und führt eine Analyse der Autokorrelation zur Frequenzbestimmung durch.
- *test\_Fourier.py* illustriert die Durchführung einer Fourier-Transformation eines periodischen Signals, das in der PicoScope-Ausgabedatei *Wellenform.csv* enthalten ist.
- *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\_mFit* nutzt das Paket *iminuit*, das den am CERN entwickelten Minimierer MINUIT nutzt. *iminuit* erlaubt insbesondere die Definition einer eigenen Kostenfunktion. Im Beispiel hier ist die Methode der kleinsten Fehlerquadrate mit Kovarianzmatrix gezeigt. Ferner unterstützt *iminuit* die Erzeugung und Darstellung von Profil-Likelihood-Kurven und Konfidenzkonturen, die im Beispiel illustriert werden. Die Behandlung von Unsicherheiten in x-Richtung ist (noch) nicht implementiert, durch Iteration und Anpassung der Kostenfunktion aber recht einfach machbar.
- *test\_kFit.py* 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\_k2Fit.py* verwendet die Version *kafe2* zur Anpassung einer Funktion an Messdaten mit unabhängigen oder korrelierten relativen oder absoluten Unsicherheiten in Ordinaten- und Abszissenrichtung.
- *test\_simplek2Fit.py* illustriert die Durchführung einer einfachen linearen Regression mit *kafe2* mit einer minimalen Anzahl eigener Codezeilen.
- *test\_Histogram.py* ist ein Beispiel zur Darstellung und statistischen Auswertung von Häufigkeitsverteilungen (Histogrammen) in ein oder 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.

Die folgenden *python*-Skripte sind etwas komplexer und zeigen typische Anwendungsfälle der Module 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, zur Suche nach Extrema und Anpassung einer Einhüllenden, zur diskreten Fourier-Transformation und zur Interpolation von Messdaten mit kubischen Spline-Funktionen.
- *Beispiel\_Hysteresse.py* demonstriert die Analyse von Daten, die mit einem USB-Oszilloskop der Marke *PicoScope* am Versuch zur Hysteresse 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 im *.labx*-Format gespeicherten Dateien ein (Beispieldatei *GammaSpektra.labx*).

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>



## MODULE DOCUMENTATION

`PhyPraKit.BuildCovarianceMatrix(sig, sigc=[])`

Construct a covariance matrix from independent and correlated error components

**Args:**

- sig: iterable of independent errors
- sigc: list of iterables of correlated uncertainties

**Returns:** covariance Matrix as numpy-array

`PhyPraKit.Cor2Cov(sig, C)`

Convert a covariance-matrix into diagonal errors + Correlation matrix

**Args:**

- V: covariance matrix as numpy array
- sigc: list of iterables of correlated uncertainties

**Returns:**

- diag uncertainties (sqrt of diagonal elements)
- C: correlation matrix as numpy array

`PhyPraKit.Cov2Cor(V)`

Convert a covariance-matrix into diagonal errors + Correlation matrix

**Args:**

- V: covariance matrix as numpy array
- sigc: list of iterables of correlated uncertainties

**Returns:**

- diag uncertainties (sqrt of diagonal elements)
- C: correlation matrix as numpy array

`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.**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.**autocorrelate**(a)

calculate autocorrelation function of input array

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

**Args:**

- a: np-array

**Returns**

- np-array of len(a), the autocorrelation function

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.**chi2p\_indep2d**(H2d, bcx, bcy, pr=True)

perform a chi2-test on independence of x and y

**method:** chi2-test on compatibility of 2d-distribution,  $f(x,y)$ , with product of marginal distributions,  $f_x(x) * f_y(y)$

**Args:**

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

**Returns:**

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

PhyPraKit.**chi2prob**(*chi2*, *ndf*)  
chi2-probability

**Args:**

- *chi2*: chi2 value
- *ndf*: number of degrees of freedom

**Returns:**

- float: chi2 probability

PhyPraKit.**convolutionEdgefinder**(*a*, *width*=10, *th*=0.0)  
find positions of maximal positive slope in data

**method:** convolute array *a* with an edge template of given width and return extrema of convoluted signal, i.e. places of rising edges

**Args:**

- *a*: array-like, input data
- *width*: int, width of signal to search for
- *th*: float, 0.<= *th* <=1., relative threshold above (global)minimum

**Returns:**

- *pidx*: list, indices (in original array) of rising edges

PhyPraKit.**convolutionFilter**(*a*, *v*, *th*=0.0)  
convolute normalized array with tmplate funtion and return maxima

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

**Args:**

- *a*: array-like, input data
- *a*: array-like, template
- *th*: float, 0.<= *th* <=1., relative threshold for places of best match above (global) minimum

**Returns:**

- *pidx*: list, indices (in original array) of best matches

PhyPraKit.**convolutionPeakfinder**(*a*, *width*=10, *th*=0.0)

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

**method:** convolute array *a* with rectangular 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, 0.<= *th* <=1., relative threshold for peaks above (global)minimum

**Returns:**

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

PhyPraKit.**generateXYdata**(*xdata, model, sx, sy, mpar=None, srelx=None, srely=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.**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.**histstat** (*binc*, *bine*, *pr=True*)

calculate mean, standard deviation and uncertainty on mean of a histogram with bin-contents *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.**k2Fit** (*func*, *x*, *y*, *sx=None*, *sy=None*, *p0=None*, *xabscor=None*, *yabscor=None*, *xrelcor=None*, *yrelcor=None*, *constraints=None*, *plot=True*, *axis\_labels=['x-data', 'y-data']*, *data\_legend='data'*, *model\_expression=None*, *model\_name=None*, *model\_legend='model'*, *model\_band='\$\pm 1 \ \sigma\$'*, *fit\_info=True*, *asym\_parerrs=True*, *plot\_cor=False*, *quiet=True*)

fit function *func* with errors on *x* and *y*; uses package *kafe2*

**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*: scalar or np-array, uncertainty(ies) on *x*
- *sy*: scalar or np-array, uncertainty(ies) on *y*
- *p0*: array-like, initial guess of parameters
- *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*
- *parameter constrains* (name, value, uncertainty)
- *plot*: flag to switch off graphical output
- *axis\_labels*: list of strings, axis labels *x* and *y*
- *data\_legend*: legend entry for data points
- *model\_name*: latex name for model function
- *model\_expression*: latex expression for model function
- *model\_legend*: legend entry for model
- *model\_band*: legend entry for model uncertainty band
- *fit\_info*: controls display of fit results on figure
- *asym\_parerrs*: show (asymmetric) errors from profile-likelihood scan
- *plot\_cor*: show profile curves and contour lines
- *quiet*: controls text output

**Returns:**

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

PhyPraKit.**kFit** (*func*, *x*, *y*, *sx=None*, *sy=None*, *p0=None*, *p0e=None*, *xabscor=None*, *yabscor=None*, *xrelcor=None*, *yrelcor=None*, *constraints=None*, *plot=True*, *title='Daten'*, *axis\_labels=['X', 'Y']*, *fit\_info=True*, *quiet=False*)

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

!!! deprecated, consider using `k2Fit()` with *kafe2* instead

**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*: scalar or np-array, uncertainty(ies) on *x*
- *sy*: scalar or np-array, 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*
- *parameter constrains* (name, value, uncertainty)
- *title*: string, title of graph
- *axis\_labels*: List of strings, axis labels *x* and *y*
- *constraints*:
- *plot*: flag to switch off graphical output
- *title*: name of data set
- *axis labels*: labels for *x* and *y* axis
- *fit info*: controls display of fit results on figure
- *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.**kRegression** (*x, y, sx, sy, xabscor=None, yabscor=None, xrelcor=None, yrelcor=None, title='Daten', axis\_labels=['x', 'y-data'], plot=True, quiet=False*)

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

!!! deprecated, consider using kFit() or k2Fit() with linear model

**Args:**

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

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

- sx: scalar or np-array, uncertainty(ies) on x
- sy: scalar or np-array, 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 graph
- axis\_labels: List of strings, axis labels x and y
- plot: flag to switch off graphical output
- 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.**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 measurement vectors
- 2d list: measurement vectors read from file

PhyPraKit.**linRegression** (*x, y, sy=None*)

linear regression  $y(x) = ax + b$

**method:** analytical formula

**Args:**

- x: np-array, independent data
- y: np-array, dependent data
- sy: scalar or 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.**linRegressionXY**(x, y, sx=None, sy=None)

linear regression  $y(x) = ax + b$  with errors on x and y uses numerical “orthogonal distance regression” from package `scipy.odr`

!!! deprecated, consider using `odFit()` with linear model instead

**Args:**

- x: np-array, independent data
- y: np-array, dependent data
- sx: scalar or np-array, uncertainty(ies) on x
- sy: scalar or np-array, uncertainty(ies) 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.**mFit**(fitf, x, y, sx=None, sy=None, erelx=None, erely=None, xabscor=None, xrelcor=None, yabscor=None, yrelcor=None, p0=None, run\_minos=True, plot=True, plot\_cor=True)

Fit an arbitrary function  $f(x)$  to data with uncorrelated and correlated absolute and/or relative errors on y and x with package `iminuit`

**mFit supports different error categories:**

- independent absolute and relative uncertainties on x and y
- correlated absolute and relative uncertainties on x and y

The fit is iterated and the covariance matrix updated dynamically during fitting to determine

- relative uncertainties with reference to model
- projection of x-uncertainties on the y-axis

**Args:**

- fitf: model function to fit, arguments (float:x, float: \*args)



- x: np-array, independent data
- y: np-array, dependent data
- sx: scalar or 1d or 2d np-array , uncertainties on x data
- sy: scalar or 1d or 2d np-array , uncertainties on x data
- erelx: scalar or np-array, relative uncertainties x
- erely: scalar or np-array, relative uncertainties y
- yabscor: scalar or np-array, absolute, correlated error(s) on y
- yrelcor: scalar or np-array, relative, correlated error(s) on y
- p0: array-like, initial guess of parameters
- run\_minos: run minos profile likelihood scan if True
- plot: show data and model if True
- plot\_cor: show profile likelihoods and confidence contours

**Returns:**

- np-array of float: parameter values
- 2d np-array of float: parameter uncertainties [0]: neg. and [1]: pos.
- np-array: correlation matrix
- float: chi2 chi-square of fit a minimum

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.**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 contents and bin edges

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.**odFit** (*fitf*, x, y, *sx=None*, *sy=None*, *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: scalar or np-array, uncertainty(ies) on x
- sy: scalar or np-array, uncertainty(ies) on y
- p0: array-like, 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.**offsetFilter** (*a*)

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

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.**readCSV** (*file*, *nlhead=1*, *delim=','*)  
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.**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.**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 fname: file name
- int ncols: number of columns
- char delimiter: character separating columns
- bool pr: print input to std out if True

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.**readtxt** (*file*, *nlhead=1*, *delim='\t'*)

**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.**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.**simplePeakfinder** (*x*, *a*, *th=0.0*)

**find positions of all maxima (peaks) 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` or `scipy.signal.argrelextrema()` instead

**Args:**

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

**Returns:**

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

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**

- abscor: absolute, correlated uncertainty
- relcor: relative, correlated uncertainty

**Returns:**

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

PhyPraKit.**wmean** (x, sx, V=None, pr=True)

**weighted mean of np-array x with uncertainties sx** or covariance matrix V; if both are given,  $sx^{**2}$  is added to the diagonal elements of the covariance matrix

**Args:**

- x: np-array of values
- sx: np-array uncertainties
- V: optional, covariance matrix of x
- pr: if True, print result

**Returns:**

- float: mean, sigma

PhyPraKit.**writeCSV** (file, ldata, hlines=[], fmt='%10g', delim=',', nline='\n', \*\*kwargs)  
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)
- delim: delimiter to separate values (default comma)
- nline: newline string

**Returns:**

- 0/1 for success/fail

PhyPraKit.**writeTextTable** (file, ldata, cnames=[], caption="", fmt='%10g')  
write data formatted as latex tabular

**Args:**

- file: string, file name
- ldata: list of columns to be written
- cnames: list of column names (optional)
- caption: LaTeX table caption (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 very general .txt formats, here the output from PicoTech 8 channel data logger, with ‘ ‘ separated values, 2 header lines, german decimal comma and special character ‘^@’

**test\_readPicoScope.py** read data exported by PicoScope usb-oscilloscope

**test\_labxParser.py** read files in xml-format produced with the Leybold Cassy system  
uses `PhyPraKit.labxParser()`

**test\_Histogram.py** demonstrate histogram functionality in `PhyPraKit`

**test\_convolutionFilter.py** Read data exported with PicoScope usb-oscilloscope, here the acoustic excitation of a steel rod

Demonstrates usage of `convolutionFilter` for detection of signal maxima and falling edges

**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\_Fourier.py** Read data exported with PicoScope usb-oscilloscope, here the acoustic excitation of a steel rod

Demonstration of a Fourier transformation of the signal

**test\_kRegression** test linear regression with `kafé` using `kFit` from `PhyPraKit` uncertainties in x and y and correlated absolute and relative uncertainties

**test\_odFit**

test fitting an arbitrary function with `scipy odr`, with uncertainties in x and y

**test\_iminuit.py** Fitting with `iminuit`

**This example illustrates the special features of `iminuit`:**

- **definition of a custom cost function** used to implement least squares method with correlated errors
- profile likelihood for asymmetric errors
- plotting of profile likelihood and confidence contours

supports `iminuit` vers. `< 2.0` and `>= 2.0`

**test\_kFit**

test fitting an arbitrary function with `kafé`, with uncertainties in x and y and correlated absolute and relative uncertainties

**test\_k2Fit**

test fitting an arbitrary function with `kafé2`, 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:**

<b>-h, --help</b>	show this help message and exit
<b>-n, --noplot</b>	suppress output of plots on screen
<b>-s, --saveplot</b>	save plot(s) in file(s)
<b>-c, --contour</b>	plot contours and profiles
<b>--noinfo</b>	suppress fit info on plot
<b>--noband</b>	suppress 1-sigma band around function
<b>--format FMT</b>	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\_Hysteresep.py

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

- 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

Beispiel\_Wellenform.py

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

- Fourier-Analyse des Signals

- Bestimmung der Resonanzfrequenz mittels Autokorrelation

Beispiel\_GammaSpektroskopie.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, [27](#)  
Beispiel\_GammaSpektroskopie, [28](#)  
Beispiel\_Hysteresese, [27](#)  
Beispiel\_Wellenform, [27](#)

### k

kfitf, [27](#)

### p

PhyPraKit, [13](#)

### t

test\_AutoCorrelation, [26](#)  
test\_convolutionFilter, [26](#)  
test\_Fourier, [26](#)  
test\_generateData, [26](#)  
test\_Histogram, [26](#)  
test\_k2Fit, [26](#)  
test\_kFit, [26](#)  
test\_kRegression, [26](#)  
test\_labxParser, [26](#)  
test\_mFit, [26](#)  
test\_odFit, [26](#)  
test\_readColumnData, [26](#)  
test\_readPicoScope, [26](#)  
test\_readtxt, [26](#)



## A

`autocorrelate()` (in module *PhyPraKit*), 14

## B

`barstat()` (in module *PhyPraKit*), 14

`Beispiel_Drehpendel`  
module, 27

`Beispiel_GammaSpektroskopie`  
module, 28

`Beispiel_Hysteresse`  
module, 27

`Beispiel_Wellenform`  
module, 27

`BuildCovarianceMatrix()` (in module  
*PhyPraKit*), 13

## C

`chi2p_indep2d()` (in module *PhyPraKit*), 14

`chi2prob()` (in module *PhyPraKit*), 14

`convolutionEdgefinder()` (in module  
*PhyPraKit*), 15

`convolutionFilter()` (in module *PhyPraKit*), 15

`convolutionPeakfinder()` (in module  
*PhyPraKit*), 15

`Cor2Cov()` (in module *PhyPraKit*), 13

`Cov2Cor()` (in module *PhyPraKit*), 13

## F

`Fourier_fft()` (in module *PhyPraKit*), 14

`FourierSpectrum()` (in module *PhyPraKit*), 13

## G

`generateXYdata()` (in module *PhyPraKit*), 15

## H

`hist2dstat()` (in module *PhyPraKit*), 16

`histstat()` (in module *PhyPraKit*), 16

## K

`k2Fit()` (in module *PhyPraKit*), 17

`kFit()` (in module *PhyPraKit*), 18

`kfitf`

module, 27

`kRegression()` (in module *PhyPraKit*), 18

## L

`labxParser()` (in module *PhyPraKit*), 19

`linRegression()` (in module *PhyPraKit*), 19

`linRegressionXY()` (in module *PhyPraKit*), 20

## M

`meanFilter()` (in module *PhyPraKit*), 21

`mFit()` (in module *PhyPraKit*), 20

module

`Beispiel_Drehpendel`, 27

`Beispiel_GammaSpektroskopie`, 28

`Beispiel_Hysteresse`, 27

`Beispiel_Wellenform`, 27

`kfitf`, 27

*PhyPraKit*, 13

`test_AutoCorrelation`, 26

`test_convolutionFilter`, 26

`test_Fourier`, 26

`test_generateData`, 26

`test_Histogram`, 26

`test_k2Fit`, 26

`test_kFit`, 26

`test_kRegression`, 26

`test_labxParser`, 26

`test_mFit`, 26

`test_odFit`, 26

`test_readColumnData`, 26

`test_readPicoScope`, 26

`test_readtxt`, 26

## N

`nhist()` (in module *PhyPraKit*), 21

`nhist2d()` (in module *PhyPraKit*), 21

## O

`odFit()` (in module *PhyPraKit*), 22

`offsetFilter()` (in module *PhyPraKit*), 22

## P

PhyPraKit  
    module, [13](#)  
profile2d() (in module *PhyPraKit*), [22](#)

## R

readCassy() (in module *PhyPraKit*), [23](#)  
readColumnData() (in module *PhyPraKit*), [23](#)  
readCSV() (in module *PhyPraKit*), [23](#)  
readPicoScope() (in module *PhyPraKit*), [23](#)  
readtxt() (in module *PhyPraKit*), [24](#)  
resample() (in module *PhyPraKit*), [24](#)

## S

simplePeakfinder() (in module *PhyPraKit*), [24](#)  
smearData() (in module *PhyPraKit*), [24](#)

## T

test\_AutoCorrelation  
    module, [26](#)  
test\_convolutionFilter  
    module, [26](#)  
test\_Fourier  
    module, [26](#)  
test\_generateData  
    module, [26](#)  
test\_Histogram  
    module, [26](#)  
test\_k2Fit  
    module, [26](#)  
test\_kFit  
    module, [26](#)  
test\_kRegression  
    module, [26](#)  
test\_labxParser  
    module, [26](#)  
test\_mFit  
    module, [26](#)  
test\_odFit  
    module, [26](#)  
test\_readColumnData  
    module, [26](#)  
test\_readPicoScope  
    module, [26](#)  
test\_readtxt  
    module, [26](#)

## W

wmean() (in module *PhyPraKit*), [25](#)  
writeCSV() (in module *PhyPraKit*), [25](#)  
writeTexTable() (in module *PhyPraKit*), [25](#)