

---

# **PhyPraKit Documentation**

***Release 1.1.3***

**Günter Quast**

**Jan 31, 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>37</b>
	<b>Index</b>	<b>39</b>



*Version* 2021-01-31



## 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 the directory *PhyPraKit* and all the files in it 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 Auswertung und Modellierung eingeführt. Die Installation der empfohlenen Software ist unter dem folgenden Link beschrieben:

- 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>

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 im Paket *PhyPraKit* enthält Funktionen zum Einlesen von Daten aus diversen Quellen, zur Datenvisualisierung, Signalbearbeitung und zur 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 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 and function fitting:
- 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\_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* 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. Dazu wird das Paket *imunit* verwendet, das den am CERN entwickelten Minimierer *MINUIT* nutzt. Da die Kostenfunktion frei definiert und auch während der Anpassung dynamisch aktualisiert werden kann, ist die Implementierung von Parameter-abhängigen Unsicherheiten möglich. Ferner unterstützt *iminuit* die Erzeugung und Darstellung von Profil-Likelihood-Kurven und Konfidenzkonturen, die so mit *mFit* ebenfalls dargestellt werden können.
- *test\_kFit.py* ist mittlerweile veraltet und dient ebenfalls 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 illustrieren 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\_Hysteresis.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 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** a collection of tools for data handling, visualisation and analysis in Physics Lab Courses, recommended for “Physikalisches Praktikum am KIT”

```
PhyPraKit.A0_readme()  
Package PhyPrakit
```

**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)
- `writeTexTable()` write data in LaTeX table format

2. signal processing:

- `offsetFilter()` subtract an offset in array a
- `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  $\chi^2$  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()` better use `matplotlib.pyplot.hist()` instead
- `histstat()` statistical information from 1d-histogram
- `nhist2d()` 2d-histogram plot based on `np.histogram2d`, `plt.colormesh()` better use `matplotlib.pyplot.hist2d()` instead
- `hist2dstat()` statistical information from 1d-histogram
- `profile2d()` “profile plot” for 2d data
- `chi2p_indep2d()`  $\chi^2$  test on independence of data

5. linear regression and function fitting:

- `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, consider using ``k2Fit`` with linear model instead
- `odFit()` fit function with x and y errors (scipy ODR)
- `mFit()` fit with iminuit with correlated x and 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

`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:**

- sig: 1d numpy array of correlated uncertainties
- C: correlation matrix as numpy array

**Returns:**

- V: covariance matrix as numpy array

`PhyPraKit.Cov2Cor(V)`

Convert a covariance-matrix into diagonal errors + Correlation matrix

**Args:**

- V: covariance matrix as numpy array

**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 implementaion for didactical purpose only, consider using `Fourier_fft()`)

**Args:**

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

**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 amplidude  $a(t)$

**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,  $0. \leq th \leq 1.$ , 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,  $0 \leq th \leq 1$ ., 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,  $0 \leq th \leq 1$ ., 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, srelx=None, srelx=None, xabscor=None, yabscor=None, xrelcor=None, yrelcor=None, ref\_to\_model=False, constraints=None, p0=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 an arbitrary function `func(x, *par)` to data points `(x, y)` with independent and correlated absolute and/or relative errors on x- and y- values with package `iminuit`.

Correlated absolute and/or relative uncertainties of input data are specified as numpy-arrays of floats; they enter in the diagonal and off-diagonal elements of the covariance matrix. Values of 0. may be specified for data points not affected by a correlated uncertainty. E.g. the array `[0., 0., 0.5., 0.5]` results in a correlated uncertainty of 0.5 of the 3rd and 4th data points. Providing lists of such array permits the construction of arbitrary covariance matrices from independent and correlated uncertainties of (groups of) data points.

**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`
- `srelx`: scalar or np-array, relative uncertainties `x`
- `srely`: scalar or np-array, relative uncertainties `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`
- `ref_to_model`, bool: refer relative errors to model if true, else use measured data
- `parameter constraints`: (name, value, uncertainty)
- `p0`: array-like, initial guess of parameters
- `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 gaph
- axis\_labels: List of strings, axis labels x and y
- parameter constraints: (name, value, uncertainty)
- 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 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*, *srelx=None*, *srely=None*, *xabscor=None*, *xrelcor=None*, *yabscor=None*, *yrelcor=None*, *p0=None*, *constraints=None*, *use\_negLogL=True*, *plot=True*, *plot\_cor=True*, *plot\_band=True*, *quiet=False*, *axis\_labels=['x', 'y = f(x, \*par)']*, *data\_legend='data'*, *model\_legend='model'*)

Fit an arbitrary function `fitf(x, *par)` to data points (*x*, *y*) with independent and correlated absolute and/or relative errors on *x*- and *y*- values with package `iminuit`.

Correlated absolute and/or relative uncertainties of input data are specified as numpy-arrays of floats; they enter in the diagonal and off-diagonal elements of the covariance matrix. Values of 0. may be specified for data points not affected by a correlated uncertainty. E.g. the array `[0., 0., 0.5., 0.5]` results in a correlated uncertainty of 0.5 of the 3rd and 4th data points. Providing lists of such arrays permits the construction of arbitrary covariance matrices from independent and correlated uncertainties of (groups of) data points.

**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 *y* data
- *srelx*: scalar or np-array, relative uncertainties *x*
- *srely*: 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
- *use\_negLogL*: use full  $-2\ln(L)$

- constraints: list or list of lists with [name or id, value, error]
- plot: show data and model if True
- plot\_cor: show profile likelihoods and confidence contours
- plot\_band: plot uncertainty band around model function
- quiet: suppress printout
- list of str: axis labels
- str: legend for data
- str: legend for model

**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='\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.**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.argrelemax()` 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**

- absco: absolute, correlated uncertainty
- relco: 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

package `iminuitFit.py`

Fitting with *iminuit* [<https://iminuit.readthedocs.io/en/stable/>]

Author: Günter Quast, initial version Jan. 2021

The code provided here supports *iminuit* vers.  $< 2$  and  $\geq 2$ .

The class *mnFit.py* uses the package *iminuit* for fitting a parameter-dependent model  $f(x, *par)$  to data points  $(x, y)$  with independent and/or correlated absolute and/or relative uncertainties in the  $x$  and/or  $y$  directions. An example function *mFit()* illustrates how to control the interface of *mnFit*, and a short script is provided to perform a fit on sample data.

**Method:** A user-defined cost function in *iminuit* with uncertainties that depend on model parameters is dynamically updated during the fitting process. Data points with relative errors can thus be referred to the model instead of the data. The derivative of the model function w.r.t.  $x$  is used to project the covariance matrix of  $x$ -uncertainties on the  $y$ -axis.

The implementation in this example is minimalistic and intended to illustrate the principle of an advanced usage of *iminuit*. It is also meant to stimulate own studies with special, user-defined cost functions.

**The main features of this package are:**

- definition of a custom cost function
- implementation of the least squares method with correlated errors
- support for correlated  $x$ -uncertainties by projection on the  $y$ -axis
- support of relative errors with reference to the model values
- evaluation of profile likelihoods to determine asymmetric uncertainties
- plotting of profile likelihood and confidence contours

The **cost function** that is optimized is a least-squares one, or an extended version if parameter-dependent uncertainties are present. In the latter case, the logarithm of the determinant of the covariance matrix is added to the least-squares cost function, so that it corresponds to twice the negative log-likelihood of a multivariate Gaussian distribution.

A fully functional example is provided by the function *mFit()* and the executable script below, which contains sample data, executes the fitting procedure and collects the results.

```
PhyPraKit.iminuitFit.mFit(fitf, x, y, sx=None, sy=None, srelx=None, srelx=None, xab-  
scor=None, xrelcor=None, yabscor=None, yrelcor=None, p0=None,  
constraints=None, use_negLogL=True, plot=True, plot_cor=True,  
plot_band=True, quiet=False, axis_labels=['x', 'y = f(x, *par)'],  
data_legend='data', model_legend='model')
```

Fit an arbitrary function  $fitf(x, *par)$  to data points  $(x, y)$  with independent and/or correlated absolute and/or relative errors on  $x$ - and/or  $y$ - values with class *mnFit* using the package *iminuit*.

Correlated absolute and/or relative uncertainties of input data are specified as floats (if all uncertainties are equal) or as numpy-arrays of floats. The concept of independent or common uncertainties of (groups) of data points is used construct the full covariance matrix from different uncertainty components. Independent uncertainties enter only in the diagonal, while correlated ones contribute to diagonal and off-diagonal elements of the covariance matrix. Values of 0. may be specified for data points not affected by a certain type of uncertainty. E.g. the array  $[0., 0., 0.5., 0.5]$  specifies uncertainties only affecting the 3rd and 4th data points. Providing lists of such arrays permits the construction of arbitrary covariance matrices from independent and correlated uncertainties of (groups of) data points.

**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
- srelx: scalar or np-array, relative uncertainties x
- srely: 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
- use\_negLogL: use full  $-2\ln(L)$
- constraints: (nested) list(s) [name or id, value, error]
- plot: show data and model if True
- plot\_cor: show profile likelihoods and confidence contours
- plot\_band: plot uncertainty band around model function
- quiet: suppress printout
- list of str: axis labels
- str: legend for data
- str: legend for model

**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

**class** PhyPraKit.iminuitFit.**mnFit**

**Fit an arbitrary function  $f(x, *par)$  to data** with independent and/or correlated absolute and/or relative uncertainties

This implementation depends on and heavily uses features of the minimizer **iminuit**.

Public methods:

- init\_data(): initialize data and uncertainties
- init\_fit(): initialize fit: data, model and parameter constraints
- setOptions(): set options for mnFit
- do\_fit(): perform fit
- plotModel(): plot model function and data
- plotContours(): plot profile likelihoods and confidence contours
- getResult(): access to final fit results
- getFunctionError(): uncertainty of model at point(s) x for parameters p
- plot\_Profile(): plot profile Likelihood for parameter

- `plot_clContour()`: plot confidence level contour for pair of parameters
- `plot_nsigContour()`: plot n-sigma contours for pair of parameters

Public data members:

- `ParameterNames`: names of parameters (as specified in model function)
- `Chi2`: chi2 at best-fit point
- `NDoF`: number of degrees of freedom
- `ParameterValues`: parameter values at best-fit point
- `MigradErrors`: symmetric uncertainties
- `CovarianceMatrix`: covariance matrix
- `CorrelationMatrix`: correlation matrix
- `OneSigInterval`: one-sigma (68% CL) ranges of parameter values
- `covx`: covariance matrix of x-data
- `covy`: covariance matrix of y-data
- `cov`: combined covariance matrix, including projected x-uncertainties

Instances of sub-classes:

- `minuit.*`: methods and members of Minuit object
- `data.*`: methods and members of sub-class `DataUncertainties`
- `costf.*`: methods and members of sub-class `xLSQ`

**static** `CL2Chi2 (CL)`

calculate DeltaChi2 from confidence level CL

**static** `Chi22CL (dc2)`

calculate confidence level CL from DeltaChi2

**class** `DataUncertainties (x, y, ex, ey, erelx, erely, cabsx, crelx, cabsy, crely, quiet=True)`

Handle data and uncertainties, build covariance matrices from components

**Args:**

- `x`: abscissa of data points (“x values”)
- `y`: ordinate of data points (“y values”)
- `ex`: independent uncertainties x
- `ey`: independent uncertainties y
- `erelx`: independent relative uncertainties x
- `erely`: independent relative uncertainties y
- `cabsx`: correlated absolute uncertainties x
- `crelx`: correlated relative uncertainties x
- `cabsy`: correlated absolute uncertainties y
- `crely`: correlated relative uncertainties y
- `quiet`: no informative printout if True

**Public methods:**

- `get_Cov()`: final covariance matrix (incl. proj. x)
- `get_xCov()`: covariance of x-values
- `get_yCov()`: covariance of y-values
- `get_iCov()`: inverse covariance matrix

**Data members:**

- copy of all input arguments
- `covx`: covariance matrix of x
- `covy`: covariance matrix of y uncertainties
- `cov`: full covariance matrix incl. projected x
- `iCov`: inverse of covariance matrix

**`get_Cov()`**

return covariance matrix of data

**`get_iCov()`**

return inverse of covariance matrix, as used in cost function

**`get_xCov()`**

return covariance matrix of x-data

**`get_yCov()`**

return covariance matrix of y-data

**`static chi2prb(chi2, ndof)`**

Calculate chi2-probability from chi2 and degrees of freedom

**`do_fit()`**

perform all necessary steps of fitting sequence

**`static getFunctionError(x, model, pvals, covp)`**

determine error of model at x

Formula:  $\Delta(x) = \sqrt{\sum_{i,j} (df/dp_i(x) df/dp_j(x) V_{p_i,j})}$

**Args:**

- x: scalar or np-array of x values
- model: model function
- pvals: parameter values
- covp: covariance matrix of parameters

**Returns:**

- model uncertainty, same length as x

**`getResult()`**

return most important results as numpy arrays

**`static get_functionSignature(f)`**

get arguments and keyword arguments passed to a function

**`init_data(x, y, ex=None, ey=1.0, erelx=None, erely=None, cabsx=None, crelx=None, cabsy=None, crely=None)`**

initialize data object

**Args:**

- x: abscissa of data points (“x values”)
- y: ordinate of data points (“y values”)
- ex: independent uncertainties x
- ey: independent uncertainties y
- erelx: independent relative uncertainties x
- erely: independent relative uncertainties y
- cabsx: correlated absolute uncertainties x
- crelx: correlated relative uncertainties x
- cabsy: correlated absolute uncertainties y
- crely: correlated relative uncertainties y
- quiet: no informative printout if True

**init\_fit** (*model*, *p0=None*, *constraints=None*)  
initialize fit object

**Args:**

- model: model function  $f(x; *par)$
- p0: np-array of floats, initial parameter values
- constraints: (nested) list(s): [parameter name, value, uncertainty] or [parameter index, value, uncertainty]

**plotContours** ()  
Plot grid of profile curves and one- and two-sigma contour lines from iminuit object

**Arg:**

- iminuitObject

**Returns:**

- matplotlib figure

**plotModel** (*axis\_labels=['x', 'y = f(x, \*par)']*, *data\_legend='data'*, *model\_legend='fit'*,  
*plot\_band=True*)  
Plot model function and data

Uses iminuitObject abd cost Fuction of type LSQwithCov

**Args:**

- list of str: axis labels
- str: legend for data
- str: legend for model

**Returns:**

- matplotlib figure

**plot\_Profile** (*pnam*)  
plot profile likelihood of parameter pnam

**plot\_clContour** (*pnam1*, *pnam2*, *cl*)  
plot a contour of parameters pnam1 and pnam2 with confidence level(s) cl

**plot\_nsigContour** (*pnam1, pnam2, nsig*)

plot nsig contours of parameters pnam1 and pnam2

**setOptions** (*relative\_refers\_to\_model=None, run\_minos=None, use\_negLogL=None, quiet=None*)

Define mnFit options

**Args:**

- rel. errors refer to model else data
- run minos else don\*t run minos
- use full neg2logL
- don\*t provide printout else verbose printout

**class xLSQ** (*data, model, quiet=True, use\_neg2logL=False*)

Custom e\_x\_tended Least-Squares cost function with dynamically updated covariance matrix and -2log(L) correction term for parameter-dependent uncertainties

For data points (x, y) with model  $f(x, *p)$  and covariance matrix  $V(f(x, *p))$  the cost function is:

$$-2 \ln \mathcal{L} = \chi^2(y, V^{-1}, f(x, *p)) + \ln(\det(V(f(x, *p))))$$

Input:

- data object of type DataUncertainties
- model function  $f(x, *par)$
- use\_neg2logL: use full -2log(L) instead of chi2 if True

`__call__` method of this class is called by iminuit

Data members:

- ndof: degrees of freedom
- nconstraints: number of parameter constraints
- chi2: chi2-value (goodness of fit)
- use\_neg2logL: usage of full 2\*neg Log Likelihood
- quiet: no printout if True

Methods:

- model(x, \*par)

**setConstraints** (*constraints*)

Add parameter constraints

format: list or list of lists of type [parameter name, value, uncertainty] or [parameter index, value, uncertainty]

**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\_readPicoSocpe.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 accoustic 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 accoustic excitation of a steel rod

Demonstraion of a Fourier transformation of the signal

**test\_kRegression** test linear regression with kafe using kFit from PhyPrakKit uncertainties in x and y and correlated absolute and relative uncertainties

**test\_odFit** test fitting an arbitrary fuction with scipy odr, with uncertainties in x and y

**test\_mFit.py** Fitting example with iminiut

Uses function PhyPraKit.mFit, which in turn uses iminuitFit

This is a rather complete example showing a fit to data with independent and correlated, absolute and relative uncertainties in the x and y directions.

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

**test\_k2Fit** Illustrate fitting of an arbitrary function with kafe2 This example illustrates the special features of kafe2: - correlated errors for x and y data - relative errors with reference to model - profile likelihood method to evaluate asymmetric errors - plotting of profile likelihood and confidence contours

**test\_generateData** 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 ouput 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\_Hysteresis.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, 34  
Beispiel\_GammaSpektroskopie, 35  
Beispiel\_Hysteresse, 35  
Beispiel\_Wellenform, 35

### k

kfitf, 34

### p

PhyPraKit, 13  
PhyPraKit.iminuitFit, 28

### t

test\_AutoCorrelation, 34  
test\_convolutionFilter, 33  
test\_Fourier, 34  
test\_generateData, 34  
test\_Histogram, 33  
test\_k2Fit, 34  
test\_kFit, 34  
test\_kRegression, 34  
test\_labxParser, 33  
test\_mFit, 34  
test\_odFit, 34  
test\_readColumnData, 33  
test\_readPicoScope, 33  
test\_readtxt, 33



## A

A0\_readme() (in module *PhyPraKit*), 13  
autocorrelate() (in module *PhyPraKit*), 15

## B

barstat() (in module *PhyPraKit*), 16  
Beispiel\_Drehpendel  
    module, 34  
Beispiel\_GammaSpektroskopie  
    module, 35  
Beispiel\_Hysteresese  
    module, 35  
Beispiel\_Wellenform  
    module, 35  
BuildCovarianceMatrix() (in module  
    *PhyPraKit*), 14

## C

Chi22CL() (PhyPraKit.iminuitFit.mnFit static  
    method), 30  
chi2p\_indep2d() (in module *PhyPraKit*), 16  
chi2prb() (PhyPraKit.iminuitFit.mnFit static  
    method), 31  
chi2prob() (in module *PhyPraKit*), 16  
CL2Chi2() (PhyPraKit.iminuitFit.mnFit static  
    method), 30  
convolutionEdgefinder() (in module  
    *PhyPraKit*), 16  
convolutionFilter() (in module *PhyPraKit*), 16  
convolutionPeakfinder() (in module  
    *PhyPraKit*), 17  
Cor2Cov() (in module *PhyPraKit*), 15  
Cov2Cor() (in module *PhyPraKit*), 15

## D

do\_fit() (PhyPraKit.iminuitFit.mnFit method), 31

## F

Fourier\_fft() (in module *PhyPraKit*), 15  
FourierSpectrum() (in module *PhyPraKit*), 15

## G

generateXYdata() (in module *PhyPraKit*), 17  
get\_Cov() (PhyPraKit.iminuitFit.mnFit.DataUncertainties  
    method), 31  
get\_functionSignature()  
    (PhyPraKit.iminuitFit.mnFit static method), 31  
get\_iCov() (PhyPraKit.iminuitFit.mnFit.DataUncertainties  
    method), 31  
get\_xCov() (PhyPraKit.iminuitFit.mnFit.DataUncertainties  
    method), 31  
get\_yCov() (PhyPraKit.iminuitFit.mnFit.DataUncertainties  
    method), 31  
getFunctionError() (PhyPraKit.iminuitFit.mnFit  
    static method), 31  
getResult() (PhyPraKit.iminuitFit.mnFit method),  
    31

## H

hist2dstat() (in module *PhyPraKit*), 18  
histstat() (in module *PhyPraKit*), 18

## I

init\_data() (PhyPraKit.iminuitFit.mnFit method),  
    31  
init\_fit() (PhyPraKit.iminuitFit.mnFit method), 32

## K

k2Fit() (in module *PhyPraKit*), 18  
kFit() (in module *PhyPraKit*), 19  
kfitf  
    module, 34  
kRegression() (in module *PhyPraKit*), 20

## L

labxParser() (in module *PhyPraKit*), 21  
linRegression() (in module *PhyPraKit*), 21  
linRegressionXY() (in module *PhyPraKit*), 22

## M

meanFilter() (in module *PhyPraKit*), 23  
mFit() (in module *PhyPraKit*), 22

`mFit()` (in module *PhyPraKit.iminuitFit*), 28  
`mnFit` (class in *PhyPraKit.iminuitFit*), 29  
`mnFit.DataUncertainties` (class  
    *PhyPraKit.iminuitFit*), 30  
`mnFit.xLSQ` (class in *PhyPraKit.iminuitFit*), 33  
module  
    *Beispiel\_Drehpendel*, 34  
    *Beispiel\_GammaSpektroskopie*, 35  
    *Beispiel\_Hysteresis*, 35  
    *Beispiel\_Wellenform*, 35  
    *kfitf*, 34  
    *PhyPraKit*, 13  
    *PhyPraKit.iminuitFit*, 28  
    *test\_AutoCorrelation*, 34  
    *test\_convolutionFilter*, 33  
    *test\_Fourier*, 34  
    *test\_generateData*, 34  
    *test\_Histogram*, 33  
    *test\_k2Fit*, 34  
    *test\_kFit*, 34  
    *test\_kRegression*, 34  
    *test\_labxParser*, 33  
    *test\_mFit*, 34  
    *test\_odFit*, 34  
    *test\_readColumnData*, 33  
    *test\_readPicoScope*, 33  
    *test\_readtxt*, 33

## N

`nhist()` (in module *PhyPraKit*), 23  
`nhist2d()` (in module *PhyPraKit*), 23

## O

`odFit()` (in module *PhyPraKit*), 24  
`offsetFilter()` (in module *PhyPraKit*), 24

## P

*PhyPraKit*  
    module, 13  
*PhyPraKit.iminuitFit*  
    module, 28  
`plot_clContour()` (*PhyPraKit.iminuitFit.mnFit*  
    method), 32  
`plot_nsigContour()` (*PhyPraKit.iminuitFit.mnFit*  
    method), 32  
`plot_Profile()` (*PhyPraKit.iminuitFit.mnFit*  
    method), 32  
`plotContours()` (*PhyPraKit.iminuitFit.mnFit*  
    method), 32  
`plotModel()` (*PhyPraKit.iminuitFit.mnFit* method),  
    32  
`profile2d()` (in module *PhyPraKit*), 24

## R

*in*  
`readCassy()` (in module *PhyPraKit*), 25  
`readColumnData()` (in module *PhyPraKit*), 25  
`readCSV()` (in module *PhyPraKit*), 24  
`readPicoScope()` (in module *PhyPraKit*), 25  
`readtxt()` (in module *PhyPraKit*), 25  
`resample()` (in module *PhyPraKit*), 26

## S

`setConstraints()` (*PhyPraKit.iminuitFit.mnFit.xLSQ*  
    method), 33  
`setOptions()` (*PhyPraKit.iminuitFit.mnFit* method),  
    33  
`simplePeakfinder()` (in module *PhyPraKit*), 26  
`smearData()` (in module *PhyPraKit*), 26

## T

*test\_AutoCorrelation*  
    module, 34  
*test\_convolutionFilter*  
    module, 33  
*test\_Fourier*  
    module, 34  
*test\_generateData*  
    module, 34  
*test\_Histogram*  
    module, 33  
*test\_k2Fit*  
    module, 34  
*test\_kFit*  
    module, 34  
*test\_kRegression*  
    module, 34  
*test\_labxParser*  
    module, 33  
*test\_mFit*  
    module, 34  
*test\_odFit*  
    module, 34  
*test\_readColumnData*  
    module, 33  
*test\_readPicoScope*  
    module, 33  
*test\_readtxt*  
    module, 33

## W

`wmean()` (in module *PhyPraKit*), 27  
`writeCSV()` (in module *PhyPraKit*), 27  
`writeTexTable()` (in module *PhyPraKit*), 27