PhyPraKit Documentation

Release 1.2.0dev

Günter Quast

May 20, 2021

CONTENTS

1 About **1**
1.1 Installation: 1

2 Visualization and Analysis of Measurement Data **3**

3 Dokumentation der Beispiele **5**

4 Module Documentation **9**

Python Module Index **37**

Index **39**

CHAPTER

ONE

ABOUT

Version 1.2.0dev, Date 2021-05-20

PhyPraKit is a collection of python modules for data visualization 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. After downloading, execute:

```
pip install --user .
```

in the main directory of the *PhyPraKit* package (where *setup.py* is located) to install in user space.

The installation via the *whl*-package provided in the subdirectory *dist* may also be used:

```
pip install --user --no-cache PhyPraKit<version>.whl
```

Installation via the PyPI Python Package Index is also available, simply execute:

```
pip install --user PhyPraKit
```

German Description:

PhyPraKit ist eine Sammlung nützlicher Funktionen in der Sprache *Python (>=3.6, die meisten Module laufen auch noch mit der inzwischen veralteten Verson 2.7)* zum Aufnehmen, zur Bearbeitung, Visualisierung und Auswertung von Daten in Praktika zur Physik. Die Anwendung der verschiedenen Funktionen des Pakets werden jeweils durch Beispiele illustriert.

CHAPTER

TWO

VISUALIZATION AND ANALYSIS OF MEASUREMENT DATA

Methods for recording, processing, visualization and analysis of measurement data are required in all laboratory courses in Physics.

This collection of tools in the package *PhyPraKit* contains functions for reading data from various sources, for data visualization, signal processing and statistical data analysis and model fitting as well as tools for generation of simulated data. Emphasis was put on simple implementations, illustrating the principles of the underlining coding.

The class *mnFit* in the module *phyFit* offers a light-weight implementation for fitting model functions to data with uncorrelated and/or correlated absolute and/or relative uncertainties in ordinate and/or abscissa directions. Support for likelihood fits to binned data (histograms) is also provided. For such complex forms of uncertainties, there are hardly any are easy-to-use program packages. Most of the existing applications use presets aiming at providing a parametrization of measurement data, whereby the validity of the parametrization is assumed and the the parameter uncertainties are scaled so that the data is well described. *PhyPraKit* offers adapted interfaces to the fit modules in the package *scipy* (*optimize.curve_fit* and *ODR*) to perform fits with a test of the validity of the hypothesis. *PhyPraKit* also contains a simplified interface to the very function-rich fitting package *kafe2*.

German: Darstellung und Auswertung von Messdaten

In allen Praktika zur Physik werden Methoden zur Aufnahme, Bearbeitung, Darstellung und Auswertung von Messdaten benötigt.

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.

Das Modul *phyFit* bietet mit der Klasse *mnFit* eine schlanke Implementierung zur Anpassung von Modellfunktionen an Daten, die mit unkorrelierten und/oder korrelierten absoluten und/oder relativen Unsicherheiten in Ordinaten- und/oder Abszissenrichtung behaftet sind. Anpassungen an gebinnte Daten (Histogramme) werden ebenfalls unterstützt. Für solche in der Physik häufig auftretenden komplexen Formen von Unsicherheiten gibt es kaum andere, einfach zu verwendende Programmpakete. Andere Pakte sind meist als Voreinstellung auf die Parametrisierung von Messdaten ausgelegt, wobei die Parameterunsicherheiten unter Annahme der Gültigkeit der Parametrisierung so skaliert werden, dass die Daten gut repräsentiert werden. *PhyPraKit* bietet entsprechend angepasste Interfaces zu den Fitmodulen im Paket *scipy* (*optimize.curve_fit* und *ODR*), um Anpassungen mit Test der Gültigkeit der Modellhypothese durchzuführen. *PhyPraKit* enthält ebenfalls ein vereinfachtes Interface zum sehr funktionsreichen Anpassungspaket *kafe2* (oder zur mittlerweile veralteten Vorgängerversion *kafe*).

In der Vorlesung "Computergestützte Datenauswertung" an der Fakultät für Physik am Karlsruher Institut für Physik (http://www.etp.kit.edu/~quast/CgDA) werden die in *PhyPraKit* verwendeten Methoden eingeführt und beschrieben. Hinweise zur Installation der empfohlenen Software finden sich unter den Links http://www.etp.kit.edu/~quast/CgDA/CgDA-SoftwareInstallation-html und http://www.etp.kit.edu/~quast/CgDA/CgDA-SoftwareInstallation.pdf

Speziell für das "Praktikum zur klassischen Physik" am KIT gibt es eine kurze Einführung in die statistischen Methoden und Werkzeuge (http://www.etp.kit.edu/~quast/CgDA/PhysPrakt/CgDA_APraktikum.pdf).

CHAPTER

THREE

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 enthält folgende **Funktionen**:

1. Daten-Ein und -Ausgabe

- readColumnData() Daten und Meta-Daten aus Textdatei lesen
- readCSV() Daten im csv-Format aus Datei mit Header lesen
- readtxt() Daten im Text-Format aus Datei mit Header lesen
- readPicoScope() mit PicoScope exportierte Daten einlesen
- readCassy() mit CASSY im .txt-Format exportierte Dateien einlesen
- labxParser() mit CASSY im .labx-Format exportierte Dateien einlesen
- writeCSV() Daten csv-Format schreiben (optional mit Header)
- writeTexTable() Daten als LaTeX-Tabelle exportieren
- round_to_error() Runden von Daten mit Präzision wie Unsicherheit
- ustring() korrekt gerundete Werte v +/- u als Text; alternativ: der Datentyp *ufloat(v, u)* im Paket *uncertainties* unterstützt die korrekte Ausgabe von Werten *v* mit Unsicherheiten *u*.

2. Signalprozessierung:

- offsetFilter() Abziehen eines off-sets
- meanFilter() gleitender Mittelwert zur Glättung
- resample() Mitteln über n Datenwerte
- simplePeakfinder() Auffinden von Maxima (Peaks) und Minima (*Empfehlung: convolutionPeakfinder nutzen*)
- convolutionPeakfinder() Finden von Maxima
- convolutionEdgefinder() Finden von Kanten
- Fourier_fft() schnelle Fourier-Transformation (FFT)
- FourierSpectrum() Fourier-Transformation *(langsam, vorzugsweise FFT-Version nutzen)*

- autocorrelate() Autokorrelation eines Signals

3. Statistik:

- wmean() Berechnen des gewichteten Mittelwerts
- BuildCovarianceMatrix() Kovarianzmatrix aus Einzelunsicherheiten
- Cov2Cor() Konversion Kovarianzmatrix -> Korrelationsmatrix
- Cor2Cov() Konversion Korrelationsmatrix + Unsicherheiten -> Kovarianzmatrix
- chi2prob() Berechnung der chi^2-Wahrscheinlichkeit
- propagatedError() Numerische Fehlerfortpflanzung; Hinweis: der Datentyp *ufloat(v, u)* im Paket *uncertainties* unterstützt Funktionen von Werten *v* mit Unsicherheiten *u* und die korrekte Fehlerfortpflanzung
- getModelError() Numerische Fehlfortpflanzung für parameterabhängige Funktionswerte

4. Histogramm:

- barstat() statistisch Information aus Histogramm (Mittelwert, Standardabweichung, Unsicherheit des Mittelwerts)
- nhist() Histogramm-Grafik mit np.historgram() und plt.bar() *(besser matplotlib.pyplot.hist() nutzen)*
- histstat() statistische Information aus 1d-Histogram
- nhist2d() 2d-Histogramm mit np.histrogram2d, plt.colormesh() *(besser matplotlib.pyplot.hist2d() nutzen)*
- hist2dstat() statistische Information aus 2d-histogram
- profile2d() "profile plot" für 2d-Streudiagramm
- chi2p_indep2d() chi^2-Test auf Unabhängigkeit zweier Variabler

5. Lineare Regression und Anpassen von Funktionen:

- linRegression() lineare Regression, y=ax+b, mit analytische Formel
- odFit() Funktionsanpassung mit x- und y-Unsicherheiten (scipy ODR)
- mFit() Funktionsanpassung mit (korrelierten) x- und y-Unsicherheiten mit *phyFit*
- hFit() Anpassung einer Verteilungsdichte an Histogramm-Daten
- k2Fit() Funktionsanpassung mit (korrelierten) x- und y-Unsicherheiten mit dem Paket *kafe2*

6. Erzeugung simulierter Daten mit MC-Methode:

- smearData() Addieren von zufälligen Unsicherheiten auf Eingabedaten
- generateXYdata() Erzeugen simulierter Datenpunkte (x+Delta_x, y+Delta_y)

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; ASCII-Sonderzeichen außer dem Spalten-Trenner werden ersetzt, ebenso wie das deutsche Dezimalkomma durch den Dezimalpunkt
- *test_readPicoScope.py* liest Ausgabedateien von USB-Oszillographen 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-Transfomation eines periodischen Signals, das in der PicoScope-Ausgabedatei *Wellenform.csv* enthalten ist.
- *test_propagatedError.py* illustriert die Anwendung von numerisch berechneter Fehlerfortpflanzung und korrekter Rundung von Größen mit Unsicherheit
- *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_hFit* illustriert die Anpassung einer Verteilungsdichte an histogrammierte Daten. Die Kostenfunktion für die Minimierung ist das zweifache der negative log-Likelihood-Funktion der Poisson-Verteilung, Poiss(k; lam), oder - optional - ihrer Annäherung durch eine Gauß-Verteilung mit Gauss(x, mu=lam, sig**2=lam). Die unsicherheiten werden aus der Modellvorhersage bestimmt, um auch Bins mit Null Einträgen korrekt behandeln zu können. Grundsätich wird eine normierte Verteilungsdichte angepasst; es ist aber optional auch möglich, die Anzahl der Einträge mit zu berücksichtigen, um die auch die Poisson-Unsicherheit der Gesamtanzahl der Histogrameinträge zu berücksichtigen.
- *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 Anzal 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.
- *toyMC_Fit.py* führt eine große Anzahl Anpassungen an simulierte Daten durch. Durch Vergleich der wahren Werte mit den aus der Anpassung bestimmten Werten lassen sich Verzerrungen der Parameterschätzungen, Korrelationen der Parameter oder die Form der Verteilung der Chi2-Wahrscheinlichkeit überprüfen, die im Idealfall eine Rechteckverteilung im Intervall [0,1] sein sollte.

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*. !!! Mittlerweile veraltet, Ersatz *kafe2go* aus dem Paket *kafe2*!

- *Beispiel_Diodenkennlinie.py* demonstriert die Analyse einer Strom-Spannungskennlinie am Beispiel von (künstlichen) Daten, an die die Shockley-Gleichung angepasst wird. Typisch für solche Messungen über einen weiten Bereich von Stromstärken ist die Änderung des Messbereichs und damit der Anzeigegenauigkeit des verwendeten Messgeräts. Im steil ansteigenden Teil der Strom-Spannungskennlinie ist es außerdem wichtig, auch die Unsicherheit der auf der x-Achse aufgetragen Spannungsmessungen zu berücksichtigen. Eine weitere Komponente der Unsicherheit ergibt sich aus der Kalibrationsgenauigkeit des Messgeräts, die als relative, korrelierte Unsicherheit aller Messwerte berücksichtigt werden muss. Das Beispiel zeigt, wie man in diesem Fall die Kovarianzmatrix aus Einzelunsicherheiten aufbaut. Die Funktionen *k2Fit()* und *mfit()* bieten dazu komfortable und leicht zu verwendende Interfaces, deren Anwendung zur Umsetzung des komplexen Fehlermodells in diesem Beispiel gezeigt wird.
- *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_Hysterese.py* demonstriert die Analyse von Daten, die mit einem USB-Oszilloskop der Marke *PicoScope* am Versuch zur Hysterese 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 Häufgkeitsverteilung 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*).

CHAPTER

FOUR

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
- round_to_error() round to same number of sigfinicant digits as uncertainty
- ustring() return rounded value +/- uncertainty as formatted string; alternative: the data type *ufloat(v,u)* of packge *uncertainties* comfortably supports printing of values *v* with uncertainties *u*.

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 calculate weighted mean
- BuildCovarianceMatrix build coraviance matrix from individual uncertainties
- Cov2Cor convert covariance matrix to correlation matrix
- Cor2Cov convert correlation matrix + errors to covariance matrix
- chi2prob caclulate chi^2 probability
- propagatedError determine propageted uncertainty, with covariance; hint: the data type *ufloat(v,u)* of packge *uncertainties* comfortably supports functions of values *v* with uncertainties *u* with correct error propagation
- getModelError determine uncertainty of parameter-depenent model function

4. histograms tools:

- barstat() statistical information (mean, sigma, error_on_mean) from bar chart
- nhist() histogram plot based on np.historgram() and plt.bar() *better use matplotlib.pyplot.hist()*
- histstat() statistical information from 1d-histogram
- nhist2d() 2d-histotram plot based on np.histrogram2d, plt.colormesh() *(better use matplotlib.pyplot.hist2d)*
- hist2dstat() statistical information from 2d-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
- odFit() fit function with x and y errors (scipy ODR)
- mFit() fit with with correlated x and y errors, profile likelihood and contour lines (module phyFit)
- hFit() fit of a density to histogram data
- k2Fit() fit function with (correlated) errors on x and y

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.<= 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.**getModelError**(*x*, *model*, *pvals*, *pcov*)
determine uncertainty of model at x from parameter uncertainties

Formula: Delta(x) = sqrt(sum_i,j (df/dp_i(x) df/dp_j(x) Vp_i,j))

Args:

- x: scalar or 1d-array of x values
- model: model function
- pvals: parameter values
- covp: covariance matrix of parameters

Returns: * model uncertainty/ies, same length as x

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*, *srely=None*, *xabscor=None*, *yabscor=None*, *xrelcor=None*, *yrelcor=None*, *ref_to_model=True*, *constraints=None*, *p0=None*, *limits=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*, *plot_band=True*, *asym_parerrs=True*, *plot_cor=False*, *showplots=True*, *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 uncertainties of (groups of) data points.

Args:

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

components of uncertainty (optional, use None if not relevant)

single float, array of length of x, or a covariance matrix

- sx: scalar, 1d or 2d np-array, uncertainty(ies) on x
- sy: scalar, 1d or 2d np-array, uncertainty(ies) on y

single float or array of length of x

- srelx: scalar or 1d np-array, relative uncertainties x
- srely: scalar or 1d np-array, relative uncertainties y

single float or array of length of x, or a list of such objects, used to construct a covariance matrix from components

- xabscor: scalar or 1d np-array, absolute, correlated error(s) on x
- yabscor: scalar or 1d np-array, absolute, correlated error(s) on y
- xrelcor: scalar or 1d np-array, relative, correlated error(s) on x
- yrelcor: scalor or 1d np-array, relative, correlated error(s) on y

fit options

- ref_to_model, bool: refer relative errors to model if true, else use measured data
- p0: array-like, initial guess of parameters
- parameter constraints: (name, value, uncertainty)
- limits: (nested) list(s) (name, min, max)

output options

- 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
- plot_band: suppress model uncertainty-band if False
- asym_parerrs: show (asymmetric) errors from profile-likelihood scan
- plot_cor: show profile curves and contour lines
- showplots: show plots on screen, default = True
- 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.**labxParser**(*file*, *prlevel=1*)
read files in xml-format produced with Leybold CASSY

Args:

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

Returns:

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

PhyPraKit.**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.**mFit** (*fitf*, *x*, *y*, *sx=None*, *sy=None*, *srelx=None*, *srely=None*, *xabscor=None*, *xrelcor=None*, *yabscor=None*, *yrelcor=None*, *ref_to_model=True*, *p0=None*, *constraints=None*, *limits=None*, *use_negLogL=True*, *plot=True*, *plot_cor=False*, *plot_band=True*, *showplots=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 class mnFit from package phyFit (uses 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 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 -2ln(L)
- constraints: (nested) list(s) [name or id, value, error]
- limits: (nested) list(s) [name or id, min, max]
- plot: show data and model if True
- plot_cor: show profile liklihoods and conficence contours
- plot_band: plot uncertainty band around model function
- showplots: show plots on screen, default = False
- 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 uncertaities [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 int(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 axix

Returns:

- float arrays: bin contents and bin edges

PhyPraKit.**nhist2d**(*x*, *y*, *bins=10*, *xlabel='x axis'*, *ylabel='y axis'*, *clabel='counts'*)
Histrogram.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 axix
- 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.**propagatedError**(*function*, *pvals*, *pcov*)
determine propageted uncertainty (with covariance matrix)

Formula: Delta = sqrt(sum_i,j (df/dp_i df/dp_j Vp_i,j))

Args:

- function: function of parameters pvals, a 1-d array is also allowed, eg. function(*p) = f(x, *p)
- pvals: parameter values
- pcov: covariance matrix (or uncertainties) of parameters

Returns:

- uncertainty Delta(function(*par))

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 fnam: 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 *int(width/2)* neighbours are averaged to from 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.**round_to_error**(*val*, *err*, *nsd_e=2*)

round float *val* to corresponding number of sigfinicant digits as uncertainty *err*

Arguments:

- val, float: value
- err, float: uncertainty of value
- nsd_e, int: number of significant digits of err

Returns:

- int: number of significant digits for v
- float: val rounded to precision of err
- float: err rounded to precision nsd_e

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.argrelmax()` 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 correspoding to peaks

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

Generate measurement data from "true" input 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: 1d np-array of floats or list of np-arrays: absolute correlated uncertainties
- relcor: 1d np-array of floats or list of np-arrays: relative correlated uncertainties

Returns:

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

PhyPraKit.**ustring**(*v*, *e*, *pe=2*)

v +/- e as formatted string with number of significant digits corresponding to precision pe of uncertainty

Args:

- v: value
- e: uncertrainty
- pe: precision (=number of significant digits) of uncertainty

Returns:

- string: <v> +/- <e> with appropriate number of digits

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 seperate values (default comma)
- nline: newline string

Returns:

- 0/1 for success/fail

PhyPraKit.**writeTexTable**(*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 phyFit.py

Physics Fitting with *iminiut* [https://iminuit.readthedocs.ios/en/stable/]

Author: Guenter Quast, initial version Jan. 2021

Requirements:

- Python >= 3.6
- iminuit vers. > 2.0
- scipy > 1.5.0
- matplotlib > 3

The class *mnFit.py* uses the optimization and uncertainty-estimation package *iminuit* for fitting a parameter-dependent model f(x, *par) to data points (x, y) or a probability density function to binned histogram data or to unbinned data. Parameter estimation is based on the Maximum-Likelihood method in the first two cases, or on a user-defined likelihood function in the latter case. Classical least-square methods are optionally available for comparison with other packages.

A unique feature of the package ist the support of different kinds of uncertainties for x-y data, namely independent and/or correlated absolute and/or relative uncertainties in the x and/or y directions. Parameter estimation for density distributions is based on the shifted Poission distibution, Poisson(x-loc, lambda), of the number of entries in each bin of a histogram.

Parameter constraints, i.e. external knowledge of parameters within Gaussian uncertainties, and limits on parameters in order to avoid problematic regions in parameter space during the minimization process, are also supported by mnFit.

Method: Uncertainties that depend on model parameters are treated by dynamically updating the cost function during the fitting process wit *iminuit*. 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.

Example functions mFit() and hFit() illustrate how to control the interface of *mnFit* for x-y and histogram fits, and a short script is provided to perform fits on sample data. A brief example how to implement fit of a prbability density to a set of (unbinned) data is also provided.

The implementation of the fitting procedure in this package is - intenitionally - rather minimalistic, and it is intended to illustrate the principle of an advanced usage of *iminuit*. It is also meant to stimulate own applications of special, user-defined cost functions.

The main features of this package are:

- provisioning of cost functions for x-y and binned histogram fits
- implementation of the least-squares method for correlated Gaussian 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 asymetric uncertainties
- plotting of profile likeliood and confidence contours

The **cost function** that is optimized for x-y fits basically is a least-squares one, which is extended 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. Fits to bistogram data rely on the negative log-likelihood of the Poisson distribution, which is generalised to support fractional observed values, which may occur if corrections to the observed bin counts have to be applied. If there is a difference between the mean value and the variance of the number of entries in a bin due to corrections, a "shifted Poisson distribution", Poiss(x-DeltaMu, lambda), is supported.

Fully functional examples are provided by the functions *mFit()* and *hFit()* and the executable script below, which contains sample data, executes the fitting procecure and collects the results. The script also contains only a few lines of code to perform a very minimalistic fit to unbinnde data with a user-defined likelihood function.

PhyPraKit.phyFit.**hFit**(*fitf*, *bin_contents*, *bin_edges*, *DeltaMu=None*, *p0=None*, *constraints=None*, *limits=None*, *use_GaussApprox=False*, *fit_density=True*, *plot=True*, *plot_cor=True*, *showplots=True*, *plot_band=True*, *quiet=False*, *axis_labels=['x', 'counts/bin = f(x, *par)']*, *data_legend='Histogram Data'*, *model_legend='Model'*)

Wrapper function to fit a density distribution f(x, *par) to binned data (histogram) with calss mnFit

The cost function is two times the negative log-likelihood of the Poission distribution, or - optionally - of the Gaussian approximation.

Uncertainties are determined from the model values in order to avoid biases and to to take account of empty bins of an histogram. The default behaviour is to fit a normalised density; optionally, it is also possible to fit the number of bin entries.

Args:

- fitf: model function to fit, arguments (float:x, float: *args)
- bin_contents:
- bin_edges:
- DeltaMu: shift mean (=mu) vs. variance (=lam), for Poisson: mu=lam
- p0: array-like, initial guess of parameters
- constraints: (nested) list(s) [name or id, value, error]
- limits: (nested) list(s) [name or id, min, max]
- GaussApprox: Gaussion approximation instead of Poisson
- density: fit density (not number of events)
- plot: show data and model if True
- plot_cor: show profile liklihoods and conficence contours

- plot_band: plot uncertainty band around model function
- showplots: show plots on screen
- quiet: suppress printout
- axis_labes: list of tow strings, axis labels
- data_legend: legend entry for data
- model_legend: legend entry for model

Returns:

- np-array of float: parameter values
- 2d np-array of float: parameter uncertaities [0]: neg. and [1]: pos.
- np-array: correlation matrix
- float: 2*negLog L, corresponding to chi-square of fit a minimum

PhyPraKit.phyFit.**mFit**(*fitf*, *x*, *y*, *sx=None*, *sy=None*, *srelx=None*, *srely=None*, *xabscor=None*, *xrelcor=None*, *yabscor=None*, *yrelcor=None*, *ref_to_model=True*, *p0=None*, *constraints=None*, *limits=None*, *use_negLogL=True*, *plot=True*, *plot_cor=True*, *showplots=True*, *plot_band=True*, *quiet=False*, *axis_labels=['x', 'y = f(x, *par)']*, *data_legend='data'*, *model_legend='model'*)

Wrapper function to 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

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. Indepenent 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 certrain 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 -2ln(L)
- constraints: (nested) list(s) [name or id, value, error]
- limits: (nested) list(s) [name or id, min, max]

- plot: show data and model if True
- plot_cor: show profile liklihoods and conficence contours
- plot_band: plot uncertainty band around model function
- showplots: show plots on screen
- 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 uncertaities [0]: neg. and [1]: pos.
- np-array: correlation matrix
- float: 2*negLog L, corresponding to chi-square of fit a minimum

class PhyPraKit.phyFit.**mnFit**(*fit_type='xy'*)

Fit an arbitrary funtion 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 and uncertainty-estimator **iminuit**.

Public Data member

- fit_type: 'xy' (default) or 'hist', controls type of fit

Public methods:

- init_data(): generic wrapper for init_*Data() methods
- init_fit(): generic wrapper for init_*Fit() methods
- setOptions(): generic wrapper for set_*Options() methods
- do_fit(): generic wrapper for do_*Fit() methods
- 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 coutour for pair of parameters
- plot_nsigContour(): plot n-sigma coutours for pair of parameters

Methods:

- init_xyData(): initialze xy data and uncertainties
- init_hData(): initialze histogram data and uncertainties
- init_xyFit(): initialize xy fit: data, model and constraints
- init_hFit(): initialize histogram fit: data, model and constraints
- init_mnFit(): initialize histogram simple minuit fit

- set_xyOptions(): set options for xy Fit
- set_hOptions(): set options for histogram Fit
- set_mnOptions(): set options for simple minuit fit with external cost function
- do_xyFit(): perform xy fit
- do_hFit(): perform histogram fit
- do_mnFit(): simple minuit fit with external, user-defined cost function

Data members:

- ParameterNames: names of parameters (as specified in model function)
- GoF: goodness-of-fit, i.e. 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 parameer 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 data sub-class, generic for xyData or hData
- costf.*: methods and members of cost sub-class, generic for xLSQ or hCost

static CL2Chi2(*CL*)
calculate DeltaChi2 from confidence level CL for 2-dim contours

static Chi22CL(*dc2*)
calculate confidence level CL from DeltaChi2 for 2-dim contours

static chi2prb(*chi2*, *ndof*)
Calculate chi2-probability from chi2 and degrees of freedom

do_hFit()
perform fit sequence for histogram fit

do_mnFit()
perform fit sequence for user-defined cost function

do_xyFit()
perform all necessary steps of fit 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) Vp_i,j))

Args:

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

Returns:

- model uncertainty, same length as x

getResult()
return most im portant results as numpy arrays

static get_functionSignature(*f*)
get arguments and keyword arguments passed to a function

class hCost(*outer*, *hData*, *model*, *use_GaussApprox=False*, *density=True*, *quiet=True*)
Cost function for binned data

The __call__ method of this class is called by iminuit.

The default cost function to minimoze is twice the negative log-likelihood of the Poisson distribution generalized to continuous observations x by replacing k! by the gamma function:

$$cost(x;\lambda) = 2\lambda(\lambda - x * \ln(\lambda) + \ln\Gamma(x+1.))$$

Alternatively, the Gaussian approximation is available:

$$cost(x;\lambda) = (x-\lambda)^2/\lambda + \ln(\lambda)$$

The implementaion also permits to shift the obervation x by an offset to take into account corrections to the number of observed bin entries (e.g. due to background or efficiency corrections): x -> x-deltaMu with deltaMu = mu - lambda, where mu is the mean of the shifted Poisson or Gauß distibution.

Input:

- outer: pointer to instance of calling class
- hData: data object of type histData
- model: model function f(x, *par)
- use_GaussApprox, bool: use Gaussian approximation
- density, bool: fit a normalised density; if false, an overall normalisation must be provided in the model function

Data members:

- ndof: degrees of freedom
- nconstraints: number of parameter constraints
- gof: goodness-of-fit as likelihood ratio w.r.t. the 'saturated model'

External references:

- model(x, *par): the model function
- data: pointer to instance of class histData
- data.model_values: bin entries calculated by the best-fit model
- data.model_related_uncertainties: uncertainties calulated from best-fit model_values

static integral_overBins (*ledges*, *redges*, *f*, **par*)
Calculate approx. integral of model over bins using Simpson's rule

static n2lLGauss (*x*, *lam*)
negative log-likelihood of Gaussian approximation Pois(x, lam) simeq Gauss(x, mu=lam, sigma^2=lam)

static n2lLPoisson (*x*, *lam*)
neg. logarithm of Poisson distribution for real-valued x

static n2lLsPoisson (*xs*, *lam*, *mu*)
2* neg. logarithm of generalized Poisson distribution: shifted to new mean mu for real-valued xk for lam=mu, the standard Poisson distribution is recovered lam=sigma*2 is the variance of the shifted Poisson distribution.

setConstraints (*constraints*)
Add parameter constraints

format: nested list(s) of type [parameter name, value, uncertainty] or [parameter index, value, uncertainty]

class histData (*outer*, *bin_contents*, *bin_edges*, *DeltaMu=None*, *quiet=True*)
Container for Histogram data

Data Members:

- contents, array of floats: bin contents
- edges, array of floats: bin edges (nbins+1 values)

calculated from input:

- nbins: number of bins
- lefts: left edges
- rights: right edges
- centers: bin centers
- widths: bin widths
- Ntot: total number of entries, used to normalize probatility density

available after completion of fit:

- model_values: bin contents from best-fit model
- model_related_uncertainties: uncertainties fom best-fit model_values

Methods:

- plot(): create figure with histogram of data and uncertainties

plot (*num='histData and Model'*, *figsize=7.5, 6.5*, *data_label='Binned data'*)
return figure with histogram data and uncertainties

init_hData (*bin_contents*, *bin_edges*, *DeltaMu=None*)
initialize histogram data object

Args: - bin_contents: array of floats - bin_edges: array of length len(bin_contents)*1 - DeltaMu: shift in mean (Delta mu) versus lambda of Poisson distribution

init_hFit (*model*, *p0=None*, *constraints=None*, *limits=None*)
initialize fit object

Args:

- model: model density 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]
- limits: (nested) list(s): [parameter name, min, max] or [parameter index, min, max]

init_mnFit(*userCostFunction*, *p0=None*, *constraints=None*, *limits=None*)
initialize fit object for simple minuit fit with user-supplied cost

Args:

- costFunction: cost function to optimize
- p0: np-array of floats, initial parameter values
- limits: (nested) list(s): [parameter name, min, max] or [parameter index, min, max]

init_xyData(*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 abolute uncertainties x
- crelx: correlated relative uncertainties x
- cabsy: correlated absolute uncertainties y
- crely: correlated relative uncertainties y
- quiet: no informative printout if True

init_xyFit(*model*, *p0=None*, *constraints=None*, *limits=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]
- limits: (nested) list(s): [parameter name, min, max] or [parameter index, min, max]

class mnCost(*outer*, *userCostFunction*, *quiet=True*)
Interface for simple minuit fit with user-supplied cost function.

The __call__ method of this class is called by iminuit.

Args:

- **userCostFunction: user-supplied cost function for minuit;** must be a negative log-likelihood

setConstraints (*constraints*)
Add parameter constraints

format: nested list(s) of type [parameter name, value, uncertainty] or [parameter index, value, uncertainty]

plotContours (*figname='Profiles and Contours'*)
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, cost Fuction (and data object)

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

static round_to_error (*val*, *err*, *nsd_e=2*)
round float *val* to same number of sigfinicant digits as uncertainty *err*

Returns:

- int: number of significant digits for v
- float: val rounded to precision of err
- float: err rounded to precision nsd_e

setLimits (*limits*)
store parameter limits

format: nested list(s) of type [parameter name, min, max] or [parameter index, min, max]

set_hOptions (*run_minos=None*, *use_GaussApprox=None*, *fit_density=None*, *quiet=None*)
Define mnFit options

Args:

- run minos else don*t run minos
- use Gaussian Approximation of Poisson distribution

- don*t provide printout else verbose printout

set_mnOptions(*run_minos=None*, *neg2logL=None*, *quiet=None*)
Define options for minuit fit with user cost function

Args:

- run_minos: run minos profile likelihood scan
- neg2logL: cost function is -2 negLogL

set_xyOptions(*relative_refers_to_model=None*, *run_minos=None*, *use_negLogL=None*, *quiet=None*)
Define options for xy fit

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(*outer*, *data*, *model*, *use_neg2logL=False*, *quiet=True*)

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:

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

For uncertainties depending on the model parameters, a more efficient approach is used to calculate the likelihood, which uses the Cholesky decompostion of the covariance matrix into a product of a triangular matrix and its transposed

$$V = LL^T,$$

thus avoiding the costy calculation of the inverse matrix.

$$\chi^2 = r \cdot (V^{-1}r) \;\; with \;\; r = y - f(x, *p)$$

is obtained by solving the linear equation

$$VX = r, \; i.e. \; X = V^{-1}r \; and \; \chi^2 = r \cdot X$$

with the effecient linear-equation solver *scipy.linalg.cho_solve(L,x)* for Cholesky-decomposed matrices.

The determinant is efficiently calculated by taking the product of the diagonal elements of the matrix L,

$$\det(V) = 2 \prod L_{i,i}$$

Input:

- outer: pointer to instance of calling class
- data: data object of type xyDataUncertainties
- model: 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
- gof: chi2-value (goodness of fit)
- use_neg2logL: usage of full 2*neg Log Likelihood
- quiet: no printpout if True

Methods:

- model(x, *par)

setConstraints(*constraints*)
Add parameter constraints

format: nested list(s) of type [parameter name, value, uncertainty] or [parameter index, value, uncertainty]

class xyDataUncertainties(*outer*, *x*, *y*, *ex*, *ey*, *erelx*, *erely*, *cabsx*, *crelx*, *cabsy*, *crely*, *quiet=True*)
Handle data and uncertainties, build covariance matrices from components

Args:

- outer: pointer to instance of calling object
- 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 abolute 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
- plot(): provide a figure with data

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

plot(*num='xyData and Model'*, *figsize=7.5, 6.5*, *data_label='data'*)
return figure with xy data and uncertainties

PhyPraKit.phyFit.**userFit**(*cost*, *p0=None*, *constraints=None*, *limits=None*, *neg2logL=True*, *plot_cor=True*, *showplots=True*, *quiet=False*)

Wrapper function to directly fit a user-defined cost funtion

This is the simplest fit possibe with the class mnFit. A user-defined cost function is minimized and an estimation of the parameter uncertainties performed

Args:

- cost: user-defined cost funtion to be minimized; the uncertaintiy estimation releys this being a negative log-likelihood function ('nlL')
- p0: array-like, initial guess of parameters
- constraints: (nested) list(s) [name or id, value, error]
- limits: (nested) list(s) [name or id, min, max]
- neg2logL: use 2 * nlL (corresponding to a least-squares-type cost)
- plot_cor: plot likelihood profiles and confidence contours of parameters
- showplots: show plots on screen (can also be done by calling script)
- quiet: contrlos verbose output

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 PhyPraPit.labxParser()

test_Historgram.py demonstrate histogram functionality in PhyPraKit

test_convolutionFilter.py Read data exported with PicoScope usb-oscilloscpe, 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-oscilloscpe, here the accoustic excitation of a steel rod

Demonstraion of a Fourier transformation of the signal

test_propagatedError.py Beispiel: Numerische Fehlerfortpflanzung mit PhyPraKit.prpagatedError() Illustriert auch die Verwendung der Rundung auf die Genauigkeit der Unsicherheit.

test_odFit test fitting an arbitrary fucntion with scipy odr, with uncertainties in x and y

test_mFit.py Fitting example with iminiut

Uses function PhyPraKit.mFit, which in turn uses mnFit from phyFit

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_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 likeliood 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

toyMC_Fit.py run a large number of fits on toyMC data to check for biases and chi2-probability distribution

This rather complete example uses eight different kinds of uncertainties, namely independent and correlated, absolute and relative ones in the x and y directions.

kfitf.py Perform a fit with the kafe package driven by input file

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

positional arguments: filename name of fit input file

optional arguments:

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

Beispiel_Diodenkennlinie.py Messung einer Strom-Spannungskennlinie und Anpassung der Schockley-Gleichung.

- Konstruktion der Kovarianzmatrix für reale Messinstrumente mit Signalrauschen, Anzeigeunsicherheiten und korrelierten, realtiven Kalibratsionsunsicherheiten für die Strom- und Spannungsmessung.
- Ausführen der Anpassung der Shockley-Gleichung mit *k2Fit* oder *mFit* aus dem Paket *PhyPraKit*. Wichtig: die Modellfunktion ist nicht nach oben beschränkt, sondern divergiert sehr schnell. Daher muss der verwendete numerische Optimierer Parameterlimits unterstützen.

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 benachberter Daten durch Mittelung
- Fourier-Transformation (einfach und fft)
- Suche nach Extrema (*peaks* und *dips*)
- Anpassung von Funkionen 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_Hysterese.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_Diodenkennlinie, 35
Beispiel_Drehpendel, 35
Beispiel_GammaSpektroskopie, 36
Beispiel_Hysterese, 36
Beispiel_Wellenform, 36

k

kfitf, 35

p

PhyPraKit, 9
PhyPraKit.phyFit, 23

t

test_AutoCorrelation, 34
test_convolutionFilter, 34
test_Fourier, 35
test_generateData, 35
test_Histogram, 34
test_k2Fit, 35
test_labxParser, 34
test_mFit, 35
test_odFit, 35
test_propagatedError, 35
test_readColumnData, 34
test_readPicoScope, 34
test_readtxt, 34
toyMC_Fit, 35

INDEX

A

A0_readme() (*in module PhyPraKit*), 9
autocorrelate() (*in module PhyPraKit*), 11

B

barstat() (*in module PhyPraKit*), 12
Beispiel_Diodenkennlinie
    module, 35
Beispiel_Drehpendel
    module, 35
Beispiel_GammaSpektroskopie
    module, 36
Beispiel_Hysterese
    module, 36
Beispiel_Wellenform
    module, 36
BuildCovarianceMatrix() (*in module PhyPraKit*), 10

C

Chi22CL() (*PhyPraKit.phyFit.mnFit static method*), 27
chi2p_indep2d() (*in module PhyPraKit*), 12
chi2prb() (*PhyPraKit.phyFit.mnFit static method*), 27
chi2prob() (*in module PhyPraKit*), 12
CL2Chi2() (*PhyPraKit.phyFit.mnFit static method*), 27
convolutionEdgefinder() (*in module PhyPraKit*), 12
convolutionFilter() (*in module PhyPraKit*), 12
convolutionPeakfinder() (*in module PhyPraKit*), 13
Cor2Cov() (*in module PhyPraKit*), 11
Cov2Cor() (*in module PhyPraKit*), 11

D

do_hFit() (*PhyPraKit.phyFit.mnFit method*), 27
do_mnFit() (*PhyPraKit.phyFit.mnFit method*), 27
do_xyFit() (*PhyPraKit.phyFit.mnFit method*), 27

F

Fourier_fft() (*in module PhyPraKit*), 11
FourierSpectrum() (*in module PhyPraKit*), 11

G

generateXYdata() (*in module PhyPraKit*), 13
get_Cov() (*PhyPraKit.phyFit.mnFit.xyDataUncertainties method*), 34
get_functionSignature() (*PhyPraKit.phyFit.mnFit static method*), 28
get_iCov() (*PhyPraKit.phyFit.mnFit.xyDataUncertainties method*), 34
get_xCov() (*PhyPraKit.phyFit.mnFit.xyDataUncertainties method*), 34
get_yCov() (*PhyPraKit.phyFit.mnFit.xyDataUncertainties method*), 34
getFunctionError() (*PhyPraKit.phyFit.mnFit static method*), 27
getModelError() (*in module PhyPraKit*), 14
getResult() (*PhyPraKit.phyFit.mnFit method*), 28

H

hFit() (*in module PhyPraKit.phyFit*), 24
hist2dstat() (*in module PhyPraKit*), 14
histstat() (*in module PhyPraKit*), 14

I

init_hData() (*PhyPraKit.phyFit.mnFit method*), 29
init_hFit() (*PhyPraKit.phyFit.mnFit method*), 29
init_mnFit() (*PhyPraKit.phyFit.mnFit method*), 30
init_xyData() (*PhyPraKit.phyFit.mnFit method*), 30
init_xyFit() (*PhyPraKit.phyFit.mnFit method*), 30
integral_overBins() (*PhyPraKit.phyFit.mnFit.hCost static method*), 28

K

k2Fit() (*in module PhyPraKit*), 14
kfitf
    module, 35

L

labxParser() (*in module PhyPraKit*), 16
linRegression() (*in module PhyPraKit*), 16

M

meanFilter() (*in module PhyPraKit*), 17
mFit() (*in module PhyPraKit*), 16
mFit() (*in module PhyPraKit.phyFit*), 25
mnFit (*class in PhyPraKit.phyFit*), 26
mnFit.hCost (*class in PhyPraKit.phyFit*), 28
mnFit.histData (*class in PhyPraKit.phyFit*), 29
mnFit.mnCost (*class in PhyPraKit.phyFit*), 30
mnFit.xLSQ (*class in PhyPraKit.phyFit*), 32
mnFit.xyDataUncertainties (*class in PhyPraKit.phyFit*), 33
module
    Beispiel_Diodenkennlinie, 35
    Beispiel_Drehpendel, 35
    Beispiel_GammaSpektroskopie, 36
    Beispiel_Hysterese, 36
    Beispiel_Wellenform, 36
    kfitf, 35
    PhyPraKit, 9
    PhyPraKit.phyFit, 23
    test_AutoCorrelation, 34
    test_convolutionFilter, 34
    test_Fourier, 35
    test_generateData, 35
    test_Histogram, 34
    test_k2Fit, 35
    test_labxParser, 34
    test_mFit, 35
    test_odFit, 35
    test_propagatedError, 35
    test_readColumnData, 34
    test_readPicoScope, 34
    test_readtxt, 34
    toyMC_Fit, 35

N

n2lLGauss() (*PhyPraKit.phyFit.mnFit.hCost static method*), 29
n2lLPoisson() (*PhyPraKit.phyFit.mnFit.hCost static method*), 29
n2lLsPoisson() (*PhyPraKit.phyFit.mnFit.hCost static method*), 29
nhist() (*in module PhyPraKit*), 18
nhist2d() (*in module PhyPraKit*), 18

O

odFit() (*in module PhyPraKit*), 18
offsetFilter() (*in module PhyPraKit*), 19

P

PhyPraKit
    module, 9
PhyPraKit.phyFit
    module, 23
plot() (*PhyPraKit.phyFit.mnFit.histData method*), 29
plot() (*PhyPraKit.phyFit.mnFit.xyDataUncertainties method*), 34
plot_clContour() (*PhyPraKit.phyFit.mnFit method*), 31
plot_nsigContour() (*PhyPraKit.phyFit.mnFit method*), 31
plot_Profile() (*PhyPraKit.phyFit.mnFit method*), 31
plotContours() (*PhyPraKit.phyFit.mnFit method*), 31
plotModel() (*PhyPraKit.phyFit.mnFit method*), 31
profile2d() (*in module PhyPraKit*), 19
propagatedError() (*in module PhyPraKit*), 19

R

readCassy() (*in module PhyPraKit*), 20
readColumnData() (*in module PhyPraKit*), 20
readCSV() (*in module PhyPraKit*), 19
readPicoScope() (*in module PhyPraKit*), 20
readtxt() (*in module PhyPraKit*), 20
resample() (*in module PhyPraKit*), 21
round_to_error() (*in module PhyPraKit*), 21
round_to_error() (*PhyPraKit.phyFit.mnFit static method*), 31

S

set_hOptions() (*PhyPraKit.phyFit.mnFit method*), 31
set_mnOptions() (*PhyPraKit.phyFit.mnFit method*), 32
set_xyOptions() (*PhyPraKit.phyFit.mnFit method*), 32
setConstraints() (*PhyPraKit.phyFit.mnFit.hCost method*), 29
setConstraints() (*PhyPraKit.phyFit.mnFit.mnCost method*), 31
setConstraints() (*PhyPraKit.phyFit.mnFit.xLSQ method*), 33
setLimits() (*PhyPraKit.phyFit.mnFit method*), 31
simplePeakfinder() (*in module PhyPraKit*), 21
smearData() (*in module PhyPraKit*), 21

T

test_AutoCorrelation
    module, 34
test_convolutionFilter
    module, 34
test_Fourier
    module, 35
test_generateData
    module, 35
test_Histogram

module, 34
test_k2Fit
module, 35
test_labxParser
module, 34
test_mFit
module, 35
test_odFit
module, 35
test_propagatedError
module, 35
test_readColumnData
module, 34
test_readPicoScope
module, 34
test_readtxt
module, 34
toyMC_Fit
module, 35

U

userFit() (*in module PhyPraKit.phyFit*), 34
ustring() (*in module PhyPraKit*), 22

W

wmean() (*in module PhyPraKit*), 22
writeCSV() (*in module PhyPraKit*), 22
writeTexTable() (*in module PhyPraKit*), 23