PhyPraKit Documentation

Release 1.1.3

Günter Quast

Feb 15, 2021

CONTENTS

Version 2021-02-15

CHAPTER ONE

ABOUT

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

Cerated 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 direcotory *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 (>=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 den physikalischen Praktika. Die Anwendung der verschiedenen Funktionen des Pakets werden jeweils durch Beispiele illustriert.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

THREE

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 Reprodzierbarkeit der Ergebnisse zu gewährleiseten.

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 foltenden 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) 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. allgemeinensten Implementierung der Vorzug gegeben.

CHAPTER

FOUR

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
- round_to_error() round to same number of sigfinicant digits as uncertainty

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 coraviance matrix
- Cov2Cor() covariance matrix to correlation matrix
- Cor2Cov() correlations + errors to covariance matrix
- chi2prob() caclulate chi^2 probability

4. histograms tools:

- barstat() statistical information (mean, sigma, erroron mean) from bar chart
- nhist() histogram plot based on np.historgram() and plt.bar() `use matplotlib.pyplot.hist() instead`
- histstat() statistical information from 1d-histogram
- nhist2d() 2d-histotram plot based on np.histrogram2d, 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() (lightweight) fit with iminuit, (correlated) uncertainties on x and y
- kFit() fit a function to data with (correlated) errors on x and y (kafe)
- k2Fit() fit a function to data with (correlated) errors on x and y (kafe2)

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-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_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 entwicklten 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 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 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*.
- *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 berücksichtig weden muss. Das Beispiel zeigt, wie man in diesem Fall die Kovarianzmatrix aus Einzelunsicherheiten aufbaut. Die Funktionen *k2Fit()* und *mfit()* bieten dazu komfortable und einfache Möglichkeiten.

- *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*).

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

CHAPTER

FIVE

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

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 coraviance matrix
- Cov2Cor covariance matrix to correlation matrix
- Cor2Cov correlations + errors to covariance matrix
- chi2prob caclulate chi^2 probability

4. histograms tools:

- barstat() statistical information (mean, sigma, erroron mean) from bar chart
- nhist() histogram plot based on np.historgram() and plt.bar() `better use matplotlib.pyplot.hist() instead`
- histstat() statistical information from 1d-histogram
- nhist2d() 2d-histotram plot based on np.histrogram2d, plt.colormesh() `better 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, 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 (kafe)
- k2Fit() fit function with (correlated) errors on x and y (kafe2)

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.`**`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*, *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*, *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

options

- 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
- 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.**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 gaph
- 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 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.`**`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*, *ref_to_model=True*, *p0=None*, *constraints=None*, *use_negLogL=True*, *plot=True*, *plot_cor=False*, *plot_band=True*, *showplots=False*, *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 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: list or list of lists with [name or id, value, error]
- 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.`**`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 d by adding random deviations according to the uncertainties

Args:

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

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

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

the following are common (correlated) systematic uncertainties

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

Returns:

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

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

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

Args:

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

Returns:

- float: mean, sigma

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

Args:

- file: string, file name

- ldata: list of columns to be written
- hlines: list with header lines (optional)
- fmt: format string (optional)
- delim: delimiter to 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 iminuitFit.py

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

Author: Guenter Quast, initial version Jan. 2021

Requirements:

- Python >= 3.6
- supports iminuit vers. < 2 and >= 2.
- scipy > 1.5.0
- matplotlib > 3

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 asymetric uncertainties
- plotting of profile likeliood 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 procecure and collects the results.

`PhyPraKit.iminuitFit.`**`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*, *use_negLogL=True*, *plot=True*, *plot_cor=True*, *showplots=False*, *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. 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]
- plot: show data and model if True
- plot_cor: show profile liklihoods and conficence 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 uncertaities [0]: neg. and [1]: pos.
- np-array: correlation matrix
- float: chi2 chi-square of fit a minimum

class PhyPraKit.iminuitFit.**mnFit**

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 **iminuit**.

Public methods:

- init_data(): initialze 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 coutour for pair of parameters
- plot_nsigContour(): plot n-sigma coutours 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 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 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 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

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) 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

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 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_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

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

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:

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

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:

- 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 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]

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_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 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 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 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

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

- Konstruktion der Kovarianzmatrix für ein reales Messinstrument
- Generierung der (simulierten) Daten
- Ausführen der Anpassung mit *mFit* aus dem Paket *iminuitFit*

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

INDEX

W