PhyPraKit Documentation

Release 1.1.2

Günter Quast

Jan 10, 2021

CONTENTS

Version 2021-01-10

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 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 file *PhyPraKit.py* in the same directory as the python scripts importing it.

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

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

1.2 Übersicht:

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

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, Darstelllung 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 Modellierung und Auswertung eingeführt.

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

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

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

Die vorliegende Sammlung von Funktionen im Paket *PhyPraKit* enthält Funktionen zum Einlesen von Daten aus diversen Quellen, zur Datenvisualisierung, Signalbearbeitung und 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
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:

- linRegression() linear regression, y=ax+b, with analytical formula
- **linRegressionXY() linear regression, y=ax+b, with x and y errors** `! deprecated, use `odFit` with linear model instead`
- **kRegression() linear regression, y=ax+b, with (correlated) errors on x and y** `! deprecated, use `kFit` or `k2Fit` with linear model instead`
- odFit() fit function with x and y errors (scipy ODR)
- mFit() fit with iminuit with correlated y errors, profile likelihood and contour lines
- kFit() fit function with (correlated) errors on x and y (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

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_kRegression.py* dient zur Anpassung einer Geraden an Messdaten mit Fehlern in Ordinaten- und Abszissenrichtung und mit allen Messpunkten gemeinsamen (d. h. korrelierten) relativen oder absoluten systematischen Fehlern mit dem Paket *kafe*.
- *test_linRegression.py* ist eine einfachere Version mit *python*-Bordmitteln zur Anpassung einer Geraden an Messdaten mit Fehlern in Ordinaten- und Abszissenrichtung. Korrelierte Unsicherheiten werden nicht unterstützt.
- *test_mFit* 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.

Die folgenden *python*-Skripte sind etwas komplexer und illustrieren typische Anwendungsfälle der Module in *PhyPraKit*:

- *kfitf.py* ist ein Kommandozeilen-Werkzeug, mit dem man komfortabel Anpassungen ausführen kann, bei denen Daten und Fit-Funktion in einer einzigen Datei angegeben werden. Beispiele finden sich in den Dateien mit der Endung *.fit*.
- *Beispiel_Drehpendel.py* demonstriert die Analyse von am Drehpendel mit CASSY aufgenommenen Daten. Enthalten sind einfache Funktionen zum Filtern und Bearbeiten der Daten, zur Suche nach Extrema und Anpassung einer Einhüllenden, zur diskreten Fourier-Transformation und zur Interpolation von Messdaten mit kubischen Spline-Funktionen.
- *Beispiel_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"

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_iminuitFit.py Fitting example with iminiut

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

kfitf.py

Perform a fit with the kafe package driven by input file

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

positional arguments: filename name of fit input file

optional arguments:

-h, --help	show this help message and exit
-n, --noplot	suppress 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_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

k

p

t

INDEX