Karlsruher Institut für Technologie

Vorlesung: Rechnernutzung in der Physik

Python, Kovarianzmatrix, Bayes

Günter Quast

Fakultät für Physik
Institut für ExperimentelleTeilchenphysik

WS 2023/24


```
# pyhton2 - python3 compatibility
from __future__ import division
from __future__ import absolute_import
from __future__ import print_function
from __future__ import unicode_literals
#
```

Enthaltene Themen:

- Bedeutung des Bayes’schen Theorems
- Details zu Python
- Kovarianz und Kovrianzmatrix

Bayes' Theorem

Bayes' Theorem

Bedingte ("conditional") Wahrscheinlichkeiten:

aus

P(A und B) = P(A) • P(B|A) = P(B) • P(A|B)

folgt

Bayes'sches Theorem:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Bayes Wahrscheinlichkeit

Frequentist-Wahrscheinlichkeit = ,,objektive" Definition

für beliebig wiederholbare Ereignisse oder bei Vohandensein von Symmetrien anwendbar

auch „Klassische Statistik" genannt

Bayes-Wahrscheinlichkeit = ,,subjektive´´ Definition

auch für einmalige Ereignisse anwendbar

Disput der Schulen zwischen **Frequentisten** und **Bayesianern** bis heute

Physiker nehmen meist einen pragmatischen Standpunkt ein – Anwendung "hybrider" Verfahren

Bedeutung von Bayes' Theorem

Besonders wichtig durch den Zusammenhang

Richtigkeit einer Theorie ↔ Beobachtung bestimmter Daten

„Likelihood“ „Prior“

$$P(Theorie|Daten) = \frac{P(Daten|Theorie)P(Theorie)}{P(Daten)}$$

„Posterior“ „Evidenz“

P(Theorie | Daten) Wahrscheinlichkeit, dass die Theorie stimmt, wenn bestimmte Daten beobachtet wurden

P(Daten | Theorie) Wahrscheinlichkeit, bestimmte Daten zu beobachten, wenn die Theorie stimmt

Interessant ist die erste Frage,
häufig wird jedoch nur die zweite beantwortet!

Beispiel: Test auf Krankheit

Wahrscheinlichkeit in allgemeiner Bevölkerung:

$$P(\text{AIDS}) = 0.001$$
$$P(\text{no AIDS}) = 0.999$$

a priori-Wissen

Ziemlich zuverlässiger AIDS-Test (Resultat + oder -):

$$P(+|\text{AIDS}) = 0.98$$
$$P(-|\text{AIDS}) = 0.02$$
$$P(+|\text{no AIDS}) = 0.03$$
$$P(-|\text{no AIDS}) = 0.97$$

Messung, Likelihoods

Wie besorgt sollte man sein, wenn man ein positives Testresultat hat?
d.h. wie groß ist (die posteriori-Wahrscheinlichkeit) P(AIDS|+) ?

Test auf Krankheit (2)

$$P(\text{AIDS}|+) = \frac{P(+|\text{AIDS})\,P(\text{AIDS})}{P(+|\text{AIDS})\,P(\text{AIDS}) + P(+|\text{no AIDS})\,P(\text{no AIDS})}$$

$$= \frac{0.98 \times 0.001}{0.98 \times 0.001 \;+\; 0.03 \times 0.999}$$

$$= 0.032$$

Die Posterior-Wahrscheinlichkeit P(AIDS|+) beträgt nur 3,2%!

Warum? Wegen der kleinen Prior-Wahrscheinlichkeit von 0.01% und der nicht vernachlässigbaren Missidentifikationswahrscheinlichkeit!

Vorsicht: Prior nicht richtig, wenn man zu einer Risikogruppe gehört!

Python

PYTHON & friends

(= numpy, scipy, matplotlib)

Literatur:

Python Einführung: www.python-kurs.eu

offizielles python Tutorial:
http://docs.python.org/3/tutorial/

numpy tutorial:
http://wiki.scipy.org/Tentative_NumPy_Tutorial

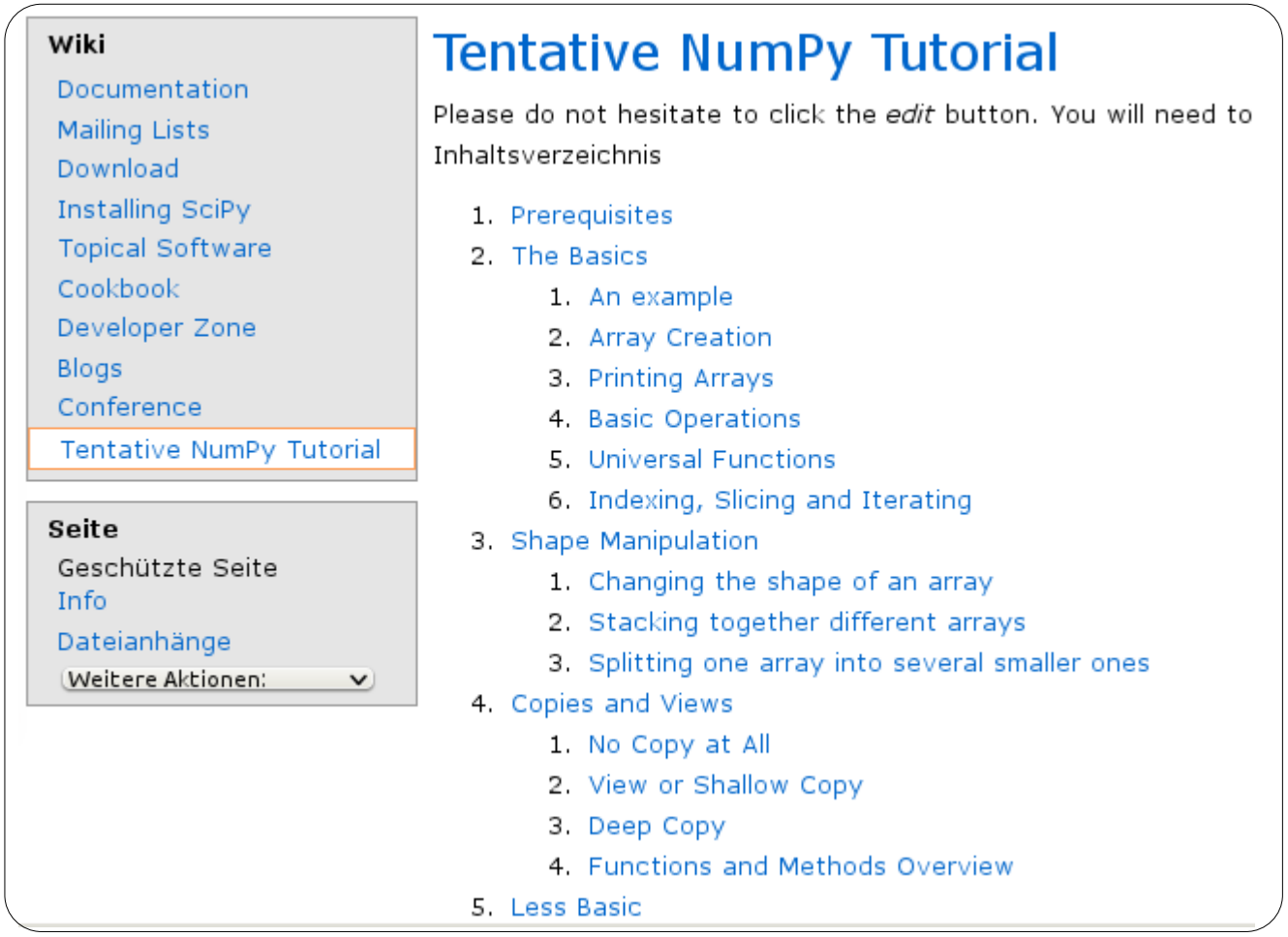
numpy reference manual
http://docs.scipy.org/doc/numpy/reference/index.html

matplotlib tutorial:
http://matplotlib.org/users/pyplot_tutorial.html

advanced – for future python programmers:
http://www.diveintopython.net/

ipython shell, an advanced interactive shell
http://ipython.org/ipython-doc/stable/interactive/tutorial.html

NumPy für Vektoren und Matrizen



Wiki

- Documentation
- Mailing Lists
- Download
- Installing SciPy
- Topical Software
- Cookbook
- Developer Zone
- Blogs
- Conference
- Tentative NumPy Tutorial

Seite

Geschützte Seite
Info
Dateianhänge
Weitere Aktionen:

Tentative NumPy Tutorial

Please do not hesitate to click the *edit* button. You will need to

Inhaltsverzeichnis

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy dimensions are called axes. The number of axes is rank.

http://www.python-kurs.eu/numpy.php

zentrale Datenstruktur: numpy array

numpy.array(

object,	„array-like“
dtype=None,	optional, Datentyp aller array-Elemente
copy=True,	wenn False, wird eine Kopie von „object“ nur wenn unbedingt notwendig erzeugt
order=None,	Anordnung der Elemente; 'A': Vorgabe, wie „object“ (auße bei Kopie, dann wie 'C'), 'C': wie in C (letzter Index läuft am schnellsten), 'F': wie in FORTRAN (1. Index läuft am schnellsten) **wichtig zur Code-Optimiereung !**
subok=False,	Vorgabe Basis-Klasse ist array, ansonsten kann eine von der Basis-Klasse von „object“ geerbt werden (z.B. von np.mat)
ndmin=0	minimale Dimension des arrays (ggf. Voranstellen von Einsen)

)¶

einfachste und häufigste Variante: a=np.array([<list>])

Beispiele zu numpy

Berechnung des gewichteten Mittelwerts

$$\bar{x} = \frac{\sum w_i \, x_i}{\sum w_i} \text{ mit } w_i = 1/{\sigma_i}^2$$

$$\sigma_{\bar{x}} = 1 \, / \sqrt{\sum w_i}$$

```
import numpy as np

w = 1/sx**2
sumw = np.sum(w)
mean = np.sum(w*x)/sumw
smean = np.sqrt(1./sumw)
```

Diskrete Fourier-Transformation

$$s_k = \frac{1}{n} \sum_i a_i \sin\left(\frac{2\pi}{f_k} t_i\right)$$

$$c_k = \frac{1}{n} \sum_i a_i \cos\left(\frac{2\pi}{f_k} t_i\right)$$

```
  import numpy as np
# input vectors a, t
  n = len(t)
  dt = (t[-1]-t[0])/(n-1.) # time step
  freq = np.linspace(0., 0.5/dt, n/2)

  omegat=np.outer(2.*np.pi*freq, t)
  s = np.matmul( np.sin(omegat), a) / n
  c = np.matmul( np.cos(omegat), a ) / n
```

Zufall aus dem Computer

numpy erlaubt es auch, Arrays (also Vektoren und Matrizen) mit **Zufallszahlen** zu füllen:

gleichverteilte Zufallszahlen

```
>>> np.random.rand()
0.466210425430829
>>> np.random.rand()
0.3937425857019299
>>> np.random.rand()
0.8586162430715967
>>> np.random.rand()
0.3181246230357016 6
>>> np.random.rand(3)
array([ 0.43266661,  0.76177195,  0.60922989])
```

gleiche Funktion, aber
jedes mal eine andere Zahl !

normal-verteilte Zufallszahlen

```
>>> np.random.randn()
0.5224185337307578
>>> np.random.randn()
1.8467581564176467
>>> np.random.randn(2)
array([-0.47682414, -0.59424037])
```

matplotlib für's Auge

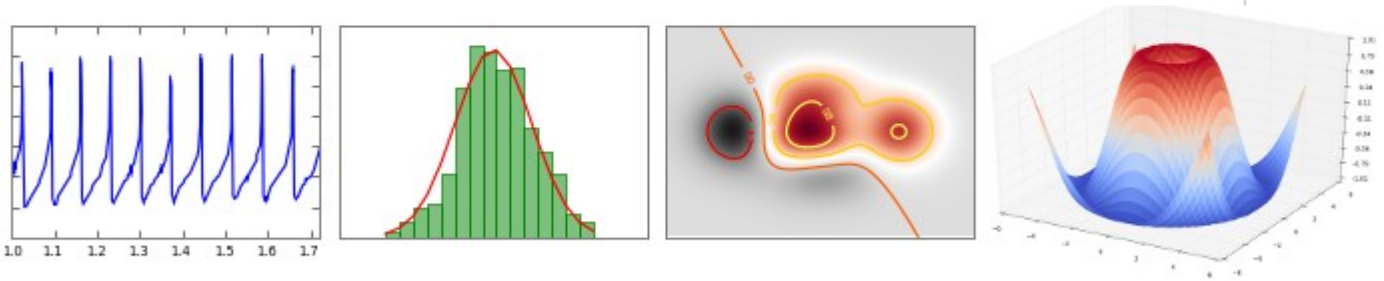
siehe http://www.matplotlib.org

matplotlib

home | examples | gallery | pyplot | docs »

Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®†), web application servers, and six graphical user interface toolkits.



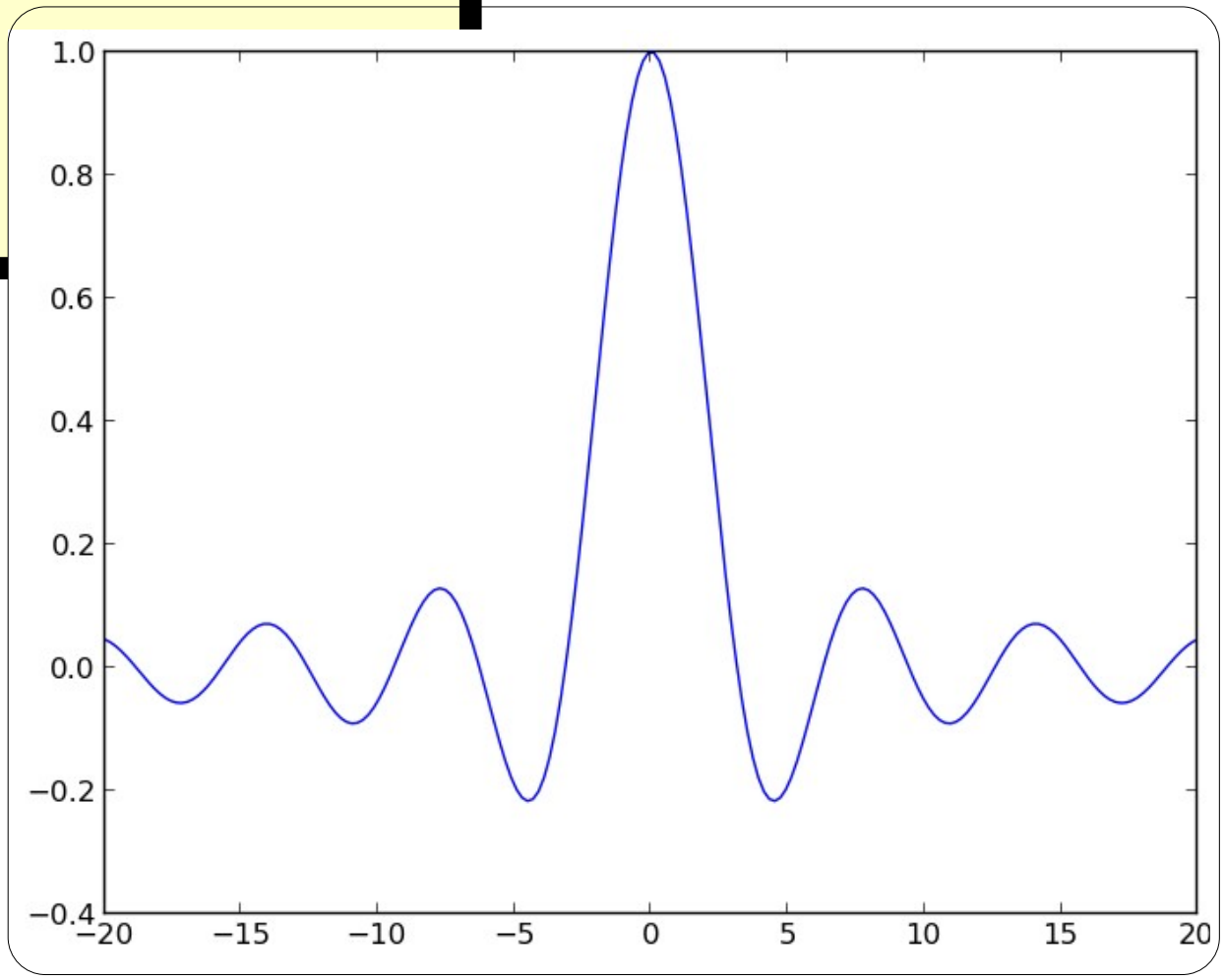
matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

Quick s
Go
Enter se
class or

matplotlib user guide: „ making plots should be easy!“

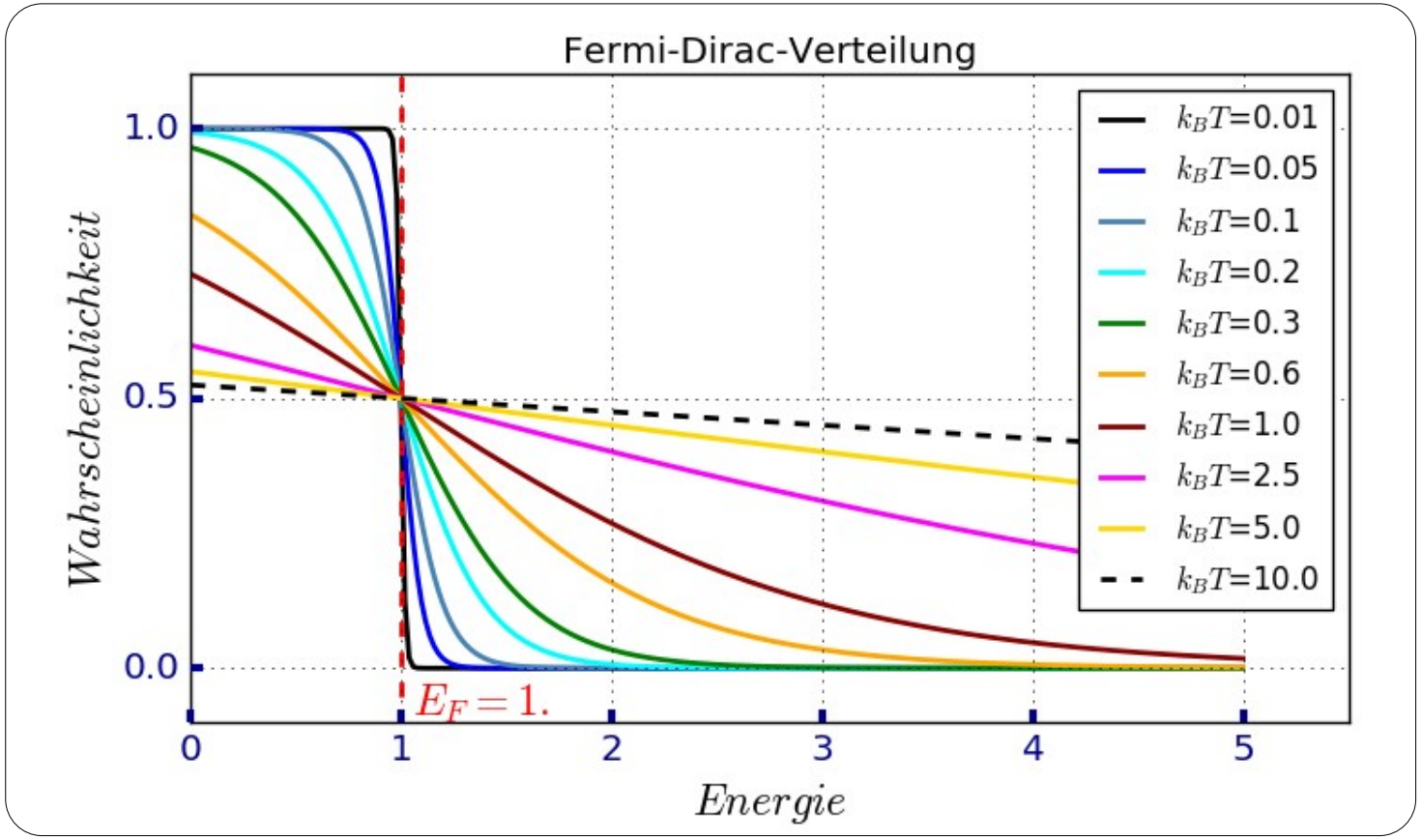
Einfaches Beispiel **numpy & matplotlib**

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>>
>>> x=np.linspace(-20,20,200)
>>> y=np.sin(x)/x
>>>
>>> plt.plot(x,y)
>>> plt.show()
```



schöne Grafiken mit `matplotlib`

Mit sehr wenig Aufwand kann man auch Beschriftungen und weitere Grafikelemente hinzufügen:



Grafische Ausgabe des Scripts `FunctionPlotter.py`

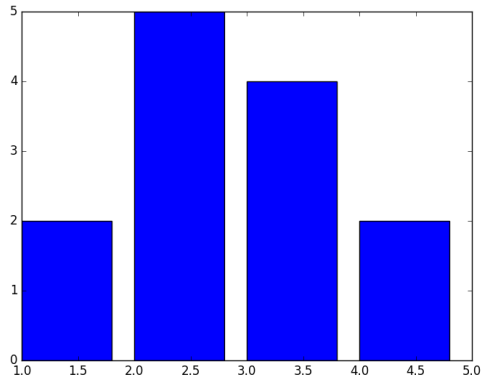
matplotlib Grundlagen

matplotlib einfügen

```
import matplotlib.pyplot as plt
```

Balkendiagramm

```
h = [3., 5., 4., 2.]
x = [1., 2., 3., 4.]
plt.bar(x, y)
```

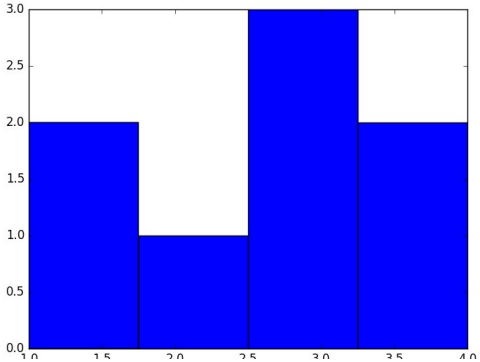


Grafik anzeigen

```
plt.show()
```

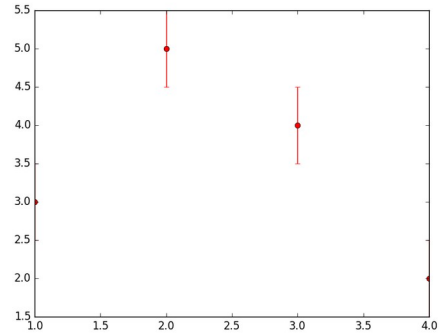
Häufigkeitsverteilung

```
a = [1, 1, 2, 3, 3, 3, 4, 4]
plt.hist(a, 4)
```



Daten mit Fehlerbalken

```
x = [1., 2., 3., 4.]
y = [3., 5., 4., 2.]
e = 0.5
plt.errorbar(x, y, yerr=e, fmt='ro')
```

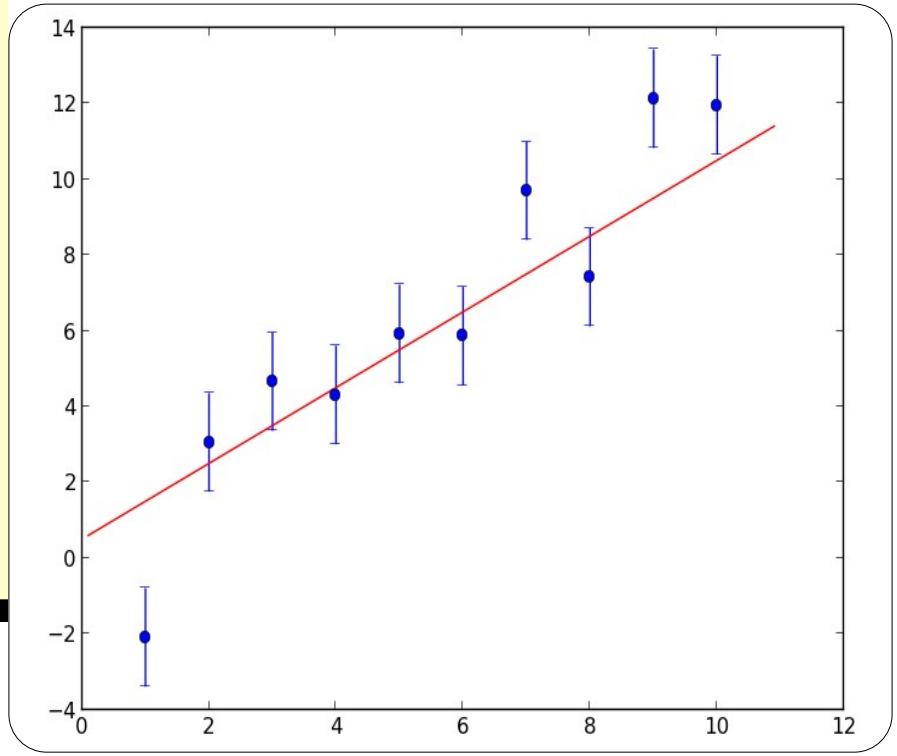


Noch nicht sehr schön, aber das kriegen wir noch ...

Falsche Daten

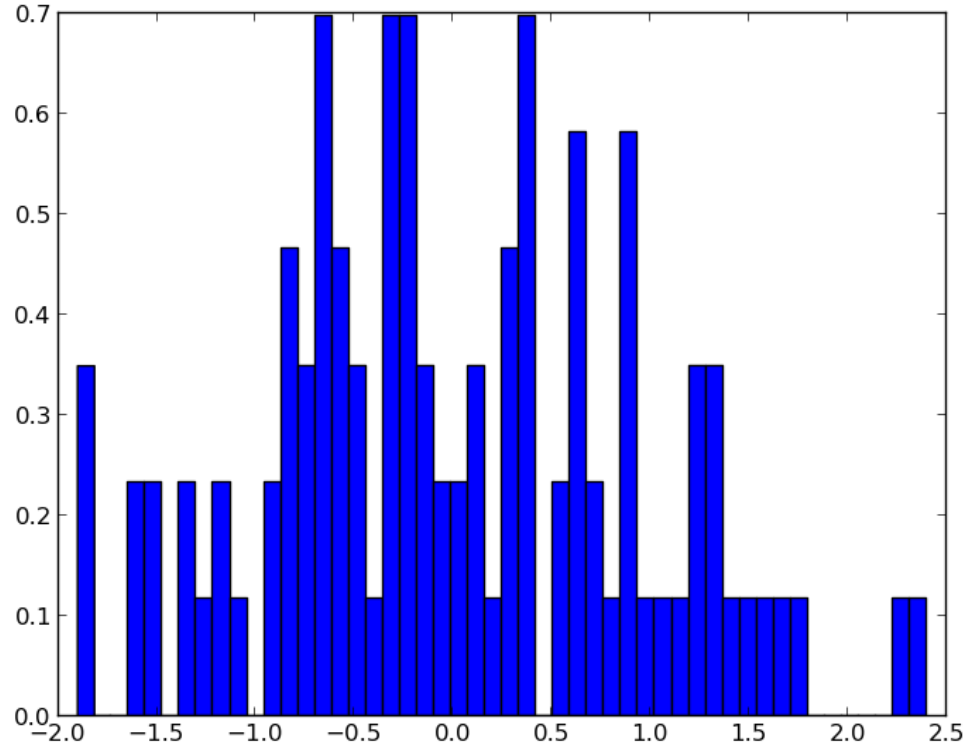
```
# Messdaten = Model + Messfehler

import numpy as np
import matplotlib.pyplot as plt

def model(x, a=1., b=0.5):
    return a*x+b

# generate fake data according to model
npoints, err =10, 1.3
xmin,xmax=1., 10.
xm=np.linspace(xmin,xmax,npoints)
dy=err*np.random.normal(size=npoints)
ym=model(xm)+dy

# plot model
dx=(xmax-xmin)/10.
x=np.linspace(xmin-dx,xmax+dx,200)
plt.plot(x,model(x),'r-')

#plot fakde data
plt.errorbar(xm,ym,yerr=err,fmt='bo')
plt.show() # display on screen
```

- **`numpy.random`** bietet Erzeugung von Zufallszahlen
- **matplotlib** (**`.pyplot.errorbar`**) erlaubt Darstellung von Datenpunkten mit Fehlerbalken



Häufigkeiten sichtbar: **Histogramm**

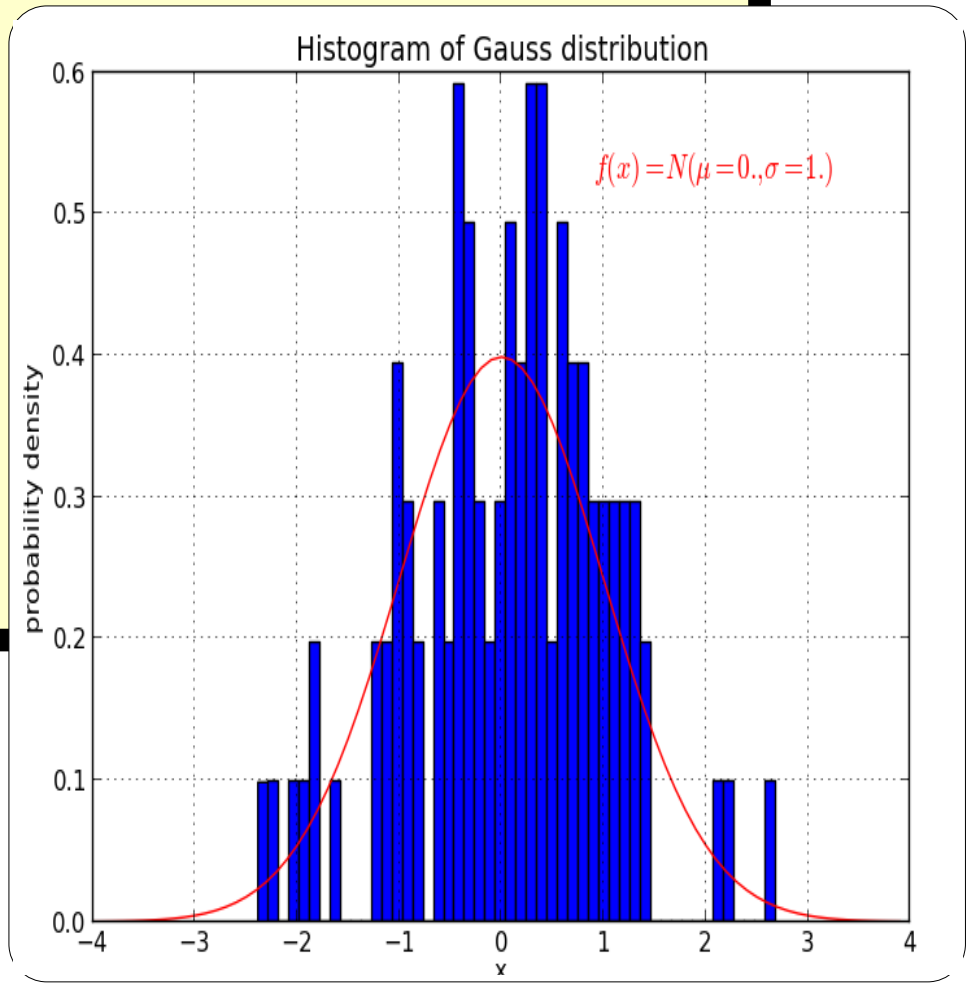
```
import numpy as np
import matplotlib.pyplot as plt

# generate random normal-distributed numbers
data=np.random.normal(size=100)

# plot data as historgram
plt.hist(data,50,normed=1)

plt.show() # put on screen
```



- **`numpy.random`**
  bietet Erzeugung von Zufallszahlen
- **`matplotlib (.pyplot.hist)`**
  bietet Erzeugung und Darstellung von Häufigkeitsverteilungen

noch schöneres Bild

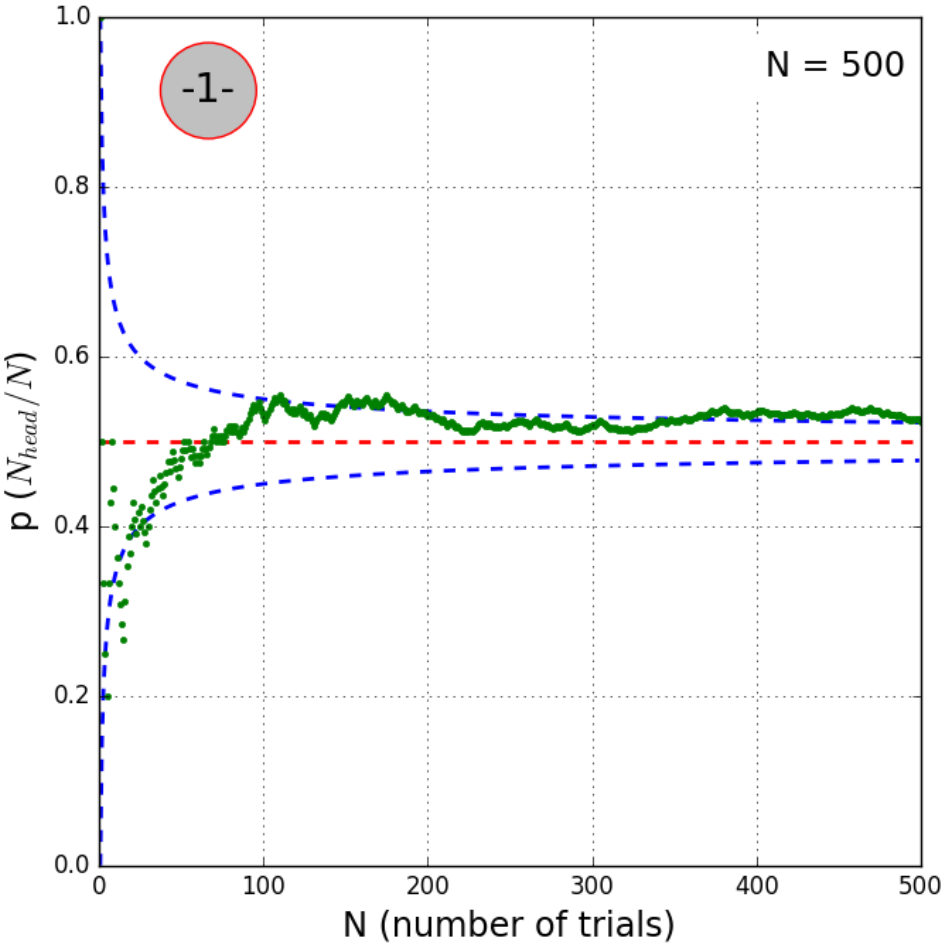
```
import numpy as np
import matplotlib.pyplot as plt

def fgauss(x,norm=1.,mu=0.,sigma=1.):    # define Gauss-function
    return (norm*np.exp(-(x-mu)**2/2/sigma**2)/np.sqrt(2*np.pi)/sigma)

data=np.random.normal(size=100) # generate Gaussian random data

#plot function and data
x = np.arange(-4., 4., 0.1)
plt.plot(x, fgauss(x), 'r-')
plt.hist(data,50,normed=1)

#     make plot nicer:
plt.xlabel('x')             # add labels
plt.ylabel('probability density')
plt.title('Gauss distribution')
plt.figtext(0.6, 0.8 \
,r'$f(x) = N(\mu=0.,\sigma=1.)$'\
,fontsize=14, color='r') # nice formula
plt.grid(True)          # show a grid
plt.show()              # on screen
```



Animierte Grafiken mit `matplotlib`

Mit Hilfe des Pakets `matplotlib.animation` können auch animierte Grafiken erstellt werden,

s. Script `animated_Coin.py`



SciPy library

Fundamental library for scientific computing

http://www.scipy.org

SciPy

enthält viele weitere nützliche Pakete für das wissenschaftliche Rechnen

spezielle
mathematische Funktionen
& Algorithmen aus den Bereichen

- Statistik
- numerische Integration
- numerische Optimierung
- Interpolation
- Signalverarbeitung (incl. FFT)
- lineare Algebra
- ...

Einschub: python-Programmierung

Spätestens wenn Sie auf dem Niveau von scipy angekommen sind, sind Ihre Programme auch für andere interessant. Sie sollten sich damit beschäftigen, wie Sie Ihren Code so schreiben und dokumentieren, so dass er für andere (und für Sie selbst) verständlich, nachvollziehbar und vertrauenswürdig ist.

Bei umfangreichen Projekten empfiehlt sich die Angabe einer Lizenz, die dafür sorgt, dass Ihr Code nur nach den dort festgelegten Regeln verwendet, weitergegeben oder verändert werden darf. Sie könnten Ihren Code z. B. als *freie Software* unter die ***Gnu Public Licence*** *(GPB)* stellen.
Dies verhindert auch, dass Sie von unbedarften Anwendern für die mit Ihrem Code erzielten Ergebnisse haftbar gemacht werden können.

Wichtige Stichpunkte:

- (aussagekräftige) Kommentarzeilen !!!
- Angabe von Autor (und evtl. Lizenzbestimmungen)
- Dokumentation, möglichst so, dass sie automatisiert aus dem Quellcode erstellt werden kann (s. google docstrings & sphinx)
- evtl. Bereitstellung einer kleinen Hilfe-Funktion
- einfaches Anwendungsbeispiel, kann auch als automatisierter Test (z.B. nach Veränderungen, Erweiterungen) genutzt werden (s. auch “unittests“)
- Unterstützung des *imports* von Funktionen in andere Programme

Special Funcions: scipy.special

```
"""scipy_example.py

Compute the maximum of a Bessel function and plot it.

Source: https://www.scipy.org/getting-started.html
"""
import argparse
import numpy as np, matplotlib.pyplot as plt
from scipy import special, optimize

def main():
    # Parse command-line arguments
    parser = argparse.ArgumentParser(usage=__doc__)
    parser.add_argument("--order", type=int, default=3,
      help="order of Bessel function")
    parser.add_argument("--output", default="plot.png",
      help="output image file")
    args = parser.parse_args()

    # Compute maximum
    f = lambda x: -special.jv(args.order, x)
    sol = optimize.minimize(f, 1.0)

    # Plot
    x = np.linspace(0, 10, 100)
    plt.plot(x, special.jv(args.order, x),
     '-', sol.x, -sol.fun, 'o')

    # Produce output
    plt.savefig(args.output, dpi=96)

if __name__ == "__main__":
    main()
```

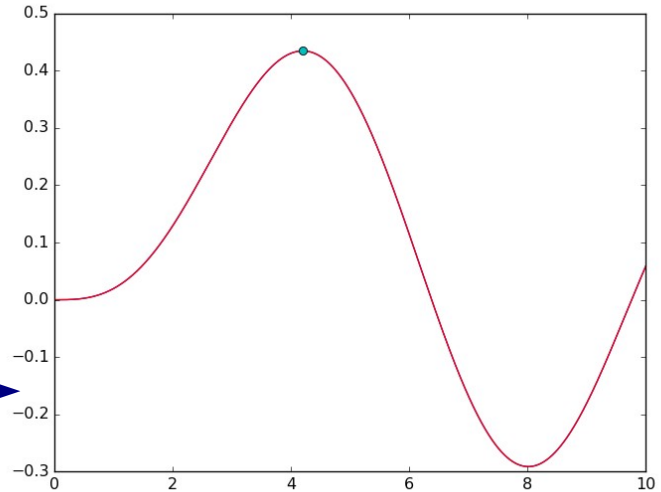
Dokumentation, kann in dieser Form z.B. mit Hilfe von sphinx in ein (html-)Dokument eingebunden werden

(schönes und professionelles) Beispielscript von http://www.scipy.org

Interface mit Hilfe für die Kommando-Zeilen

Code ab hier wird bei import nicht ausgeführt (nützlich für Test oder Anwendungsbeispiel)

Suche des Maximums einer Bessel-Funktion und grafische Darstellung



Differentialgleichungen lösen mit scipy

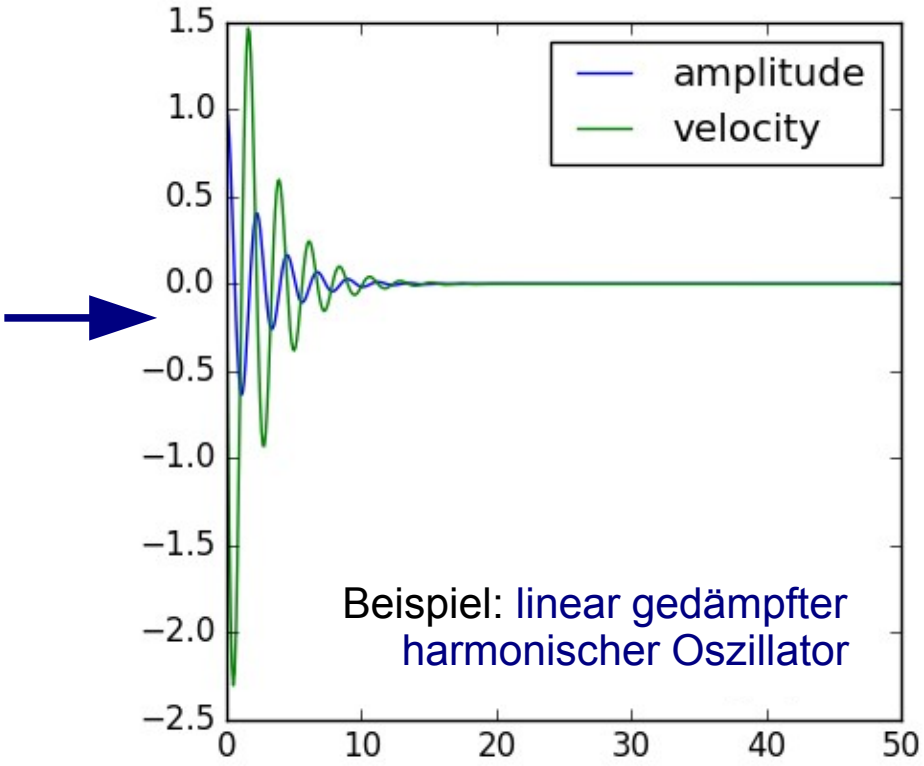
```
import numpy as np, matplotlib.pyplot as plt
from scipy.integrate import odeint

mass = 0.5
kspring = 4
cdamp = 0.4

nu_coef = cdamp / mass
om_coef = kspring / mass

def deriv_lindamp(yvec, time, nu, omc):
  ''' calculate derivatives: x´=v,  v´=x´´  '''
    return (yvec[1], -nu * yvec[1] - omc * yvec[0])

# parameters for numerical solution
tmax=50.  # time span
dt=0.1    # time step
time_vec = np.linspace(0, tmax, int(tmax/dt))

# initial conditions
a0=1.
v0=0.
# get solution vector from odeint
arr_lindamp = odeint(deriv_lindamp, (a0,v0),
    time_vec, args=(nu_coef, om_coef))

fig=plt.figure(figsize=(10., 10.))
ax=fig.add_subplot(2,2,1)
ax.text(0.5, 0.05, 'linear damping', transform=ax.transAxes)
ax.plot(time_vec, arr_lindamp[:, 0], label='amplitude')
ax.plot(time_vec, arr_lindamp[:, 1], label="velocity")
ax.legend()
plt.show()
```

Auszug aus Script
odeint_scillators.py

Paket `scipy.odeint`

löst Systeme von Differentialgleichungen erster Ordnung numerisch.

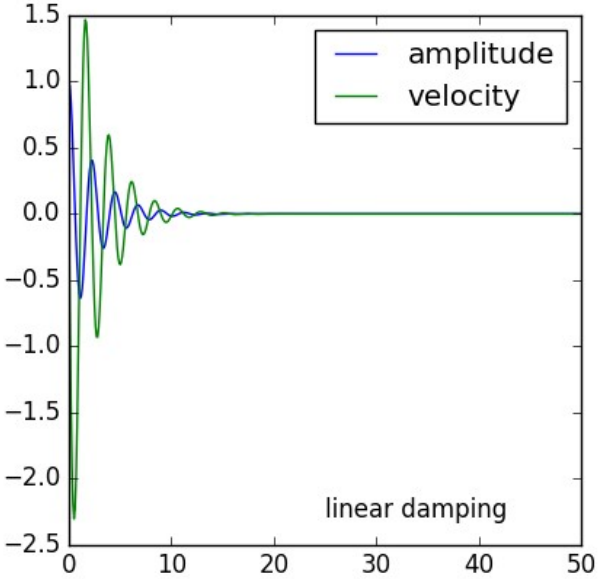
Differentialgleichungen höherer Ordnung werden auf ein System von Gleichungen 1. Ordnung transfc $\dot{x} =: v \,, \rightarrow \ddot{x} = \dot{v}$

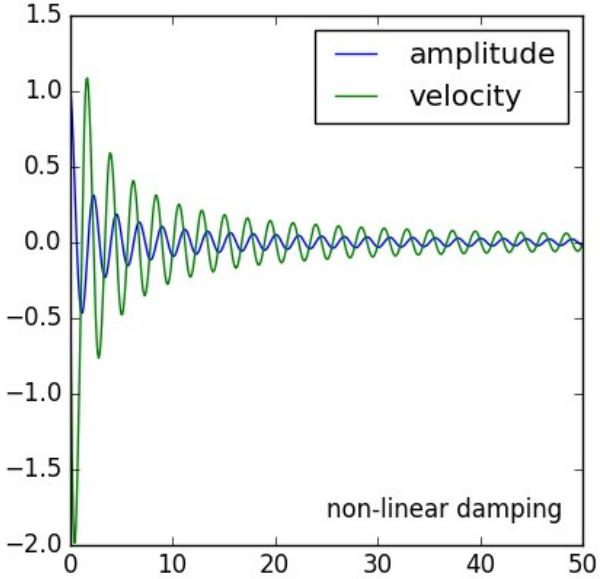


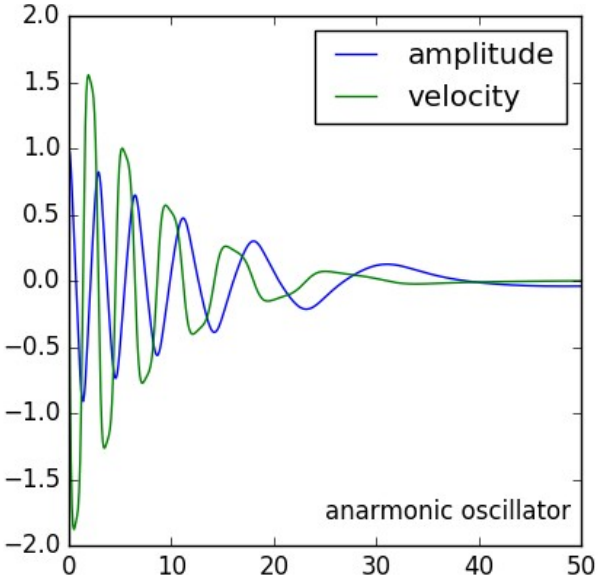
Differentialgleichungen lösen mit scipy (2)

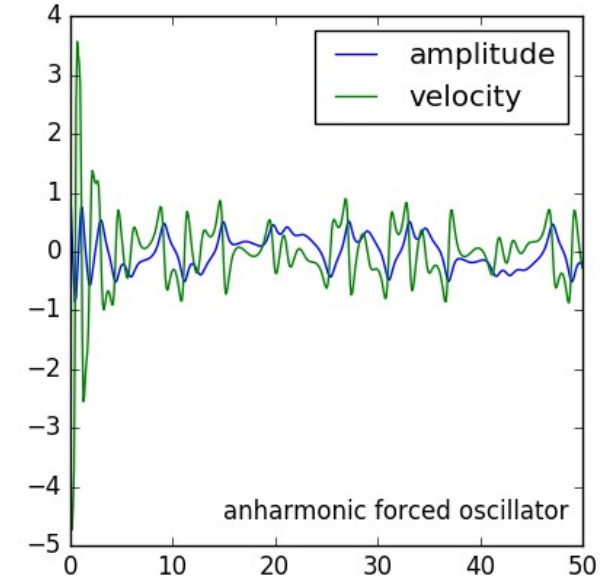
Paket `scipy.odeint`

Script
`odeint_oscillators.py`







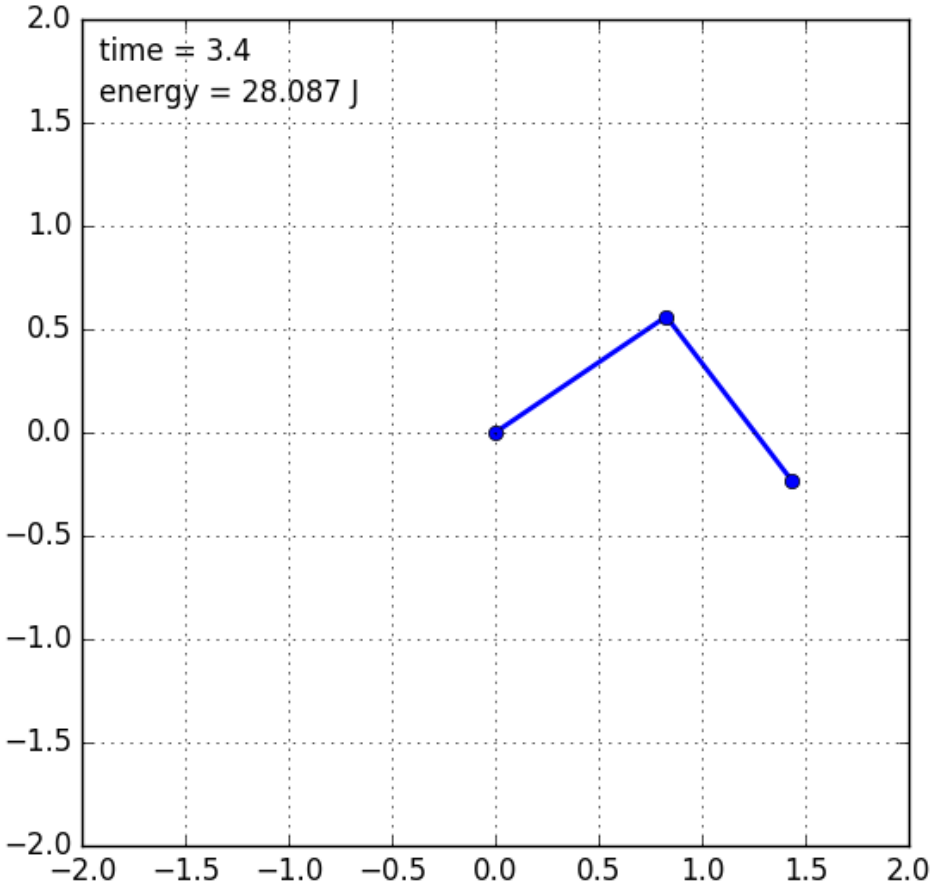


Wenn man die Differential-Gleichung ändert, löst der gleiche Code die Bewegungsgleichungen für alle möglichen Arten von (nicht-linearen) Oszillatoren.

`scipy.odeint` in Beispiel mit Animation

mit recht wenig Aufwand funktioniert auch das Lösen von Differentialgleichungssystemen in Echtzeit, z. B. das aus Theorie B bekannte (chaotische) Doppelpendel. Mit dem Paket `matplotlib.animation` lässt sich das auch als animierte Grafik darstellen:

`s. Script double_pendulum.py`



Beispiel-Code

Alle Beispiele dieser Vorlesung finden Sie im Ordner **V02Beispiele**

Eine Sammlung von Beispielen speziell für die Anwendung in den Praktika enthält das Paket **PhyPraKit**

http://etp.kit.edu/~quast/PhyPraKit/

python-Modul PhyPraKit.py

1. Einlesen von Daten aus Text-Dateien
   - read_data() n Spalten mit Meta-Daten (Datum, Autor, …)
   - read_csv() Daten als „Comma Separated Values“
   - read_txt() allg. Daten im Text-Format
2. Methoden zur Prozessierung von Roh-Daten
   Glätten, Suche nach Extrema und Flanken, Fouriertransformation, Autokorrelation
3. Berechnung des gewichteten Mittelwerts
4. Histogramm-Tools
   - barstat() statistische Information aus Balkendiagramm
   - histstat() statistiche information aus 1d-Histogramm
   - hist2dstat() statistische Information aus 2d-histogram
   - profile2d() "profile plot" für 2d-Daten
   - chi2p_indep2d() chi2-Test auf Unabhängigkeit zweier Datensätze
5. lineare Regression `(y = a · x + b)`, Funktionsanpassung
   - linRegression() mit der analytischen Formel (nur y-Fehler)
   - kFit() numerisch mit `kafe`, x-, y- und korrelierte Fehler
6. Erzeugung von simulierten Datensätzen

Paket phyFit: Anpassungen von Modellen an Messdaten mit **iminuit**

s. insb. Beispiel-Scripts

```
test_readtxt.py

test_Fourier
test_AutoCorrelation.py

test_Histogram.py

test_simplek2Fit.py
test_k2Fit.py
test_xyFit.py
test_mlFit.py
test_hFit.py

test_generateData.py
```

Es gibt noch viel zu entdecken !

Einfach ausprobieren ...

Beste Methode:
- Beispiele
  - analysieren,
    - für eigene Zwecke anpassen,
      - weiterentwickeln.

Die Kovarianzmatrix

Kovarianz - der Zusammenhang zwischen Variablen

Kovarianz zweier Zufallsvariablen ist **Erwartungswert** von

```
(Abweichung vom Erwartungswert in Variable x) *
(Abweichung vom Erwartungswert in Variable y)
```

$$\mathrm{cov}(x,y) = E[(x-\mu_x)(y-\mu_y)] = \ldots = E[xy] - \mu_x\mu_y$$

Analog auch bei mehr als zwei Variablen:

cov(x_i, x_j) sind die Elemente der **Kovarianzmatrix V**

Diagonalwerte sind die Varianzen !

Anm.: für mehrere Variable $x_i = (\vec{x})_i$ kann man die Kovarianzmatrix auch sehr kompakt in Matrix-Schreibweise darstellen:

$$\mathbf{V} = E\left[(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^T\right] = E\left[\vec{x}\vec{x}^T\right] - \vec{\mu}\vec{\mu}^T$$

das ist formal identisch zur Definition der Varianz:
statt Produkt zweier Zahlen das „dyadische Produkt" von Vektoren

gemeinsame Unsicherheiten & Kovarianz

Erinnerung: $m = w + z$

Messwert = wahrer Wert + Zufallskomponente

Wir betrachten nun mehrere (n) Messwerte m_i :
jede Messung hat
- eine individuelle, zufällige Unsicherheit z_i ($E[z_i]= 0$)
- sowie eine gemeinsame Unsicherheit z_g ($E[z_g]= 0$)

$\rightarrow$ $m_i = w + z_i + z_g$

ein gemeinsamer wahrer Wert und n+1 Zufallszahlen
(eine allen Messungen gemeinsame sowie n individuelle)

Beispiel zwei Messungen:

$$
\begin{aligned}
\mathrm{cov}_{1,2} &= E\left[(m_1-\mu_1)(m_2-\mu_2)\right] \\
&= E\left[(w+z_1+z_g-\mu)(w+z_2+z_g-\mu)\right] \\
&= E\left[(\mu+z_1+z_g-\mu)(\mu+z_2+z_g-\mu)\right] \\
&= E\left[z_1+z_g)(z_2+z_g)\right] \\
&= E\left[z_1z_2+z_1z_g+z_gz_2+z_gz_g)\right] \\
&= E[z_1z_2]+E[z_1z_g]+E[z_gz_2]+E[z_gz_g] \\
& \quad z_1, z_2 \text{ und } z_g \text{ sind unabhängig} \\
&= 0+0+0+E[z_gz_g] \\
&= V(z_g)=\sigma_g{}^2
\end{aligned}
$$

Das Kovarianzmatrixelement ist
das Quadrat der gemeinsamen Unsicherheit der Messungen !

Kovarianzmatrix von korrelierten Messungen

Mehrere (n) Messwerte m_i , jede Messung hat
- eine individuelle, zufällige Unsicherheit z_i
- sowie eine gemeinsame Unsicherheit z_g

$\rightarrow$ $m_i = w + z_i + z_g$

ein gemeinsamer wahrer Wert und n+1 Zufallszahlen
(eine allen Messungen gemeinsame sowie n individuelle)

Für n Messwerte m_i erhält man die Kovarianz-Matrix, indem man die Varianz der gemeinsamen Unscherheit z_g, $(\sigma_g)^2$, in die Nebendiagonale setzt. Die Diagonalelemente sind die Varianzen der Gesamtunsicherheiten, $(\sigma_i)^2 + (\sigma_g)^2$.

Die vollständige Kovarianzmatrix sieht so aus:

$$V = \begin{pmatrix} {\sigma_1}^2 + {\sigma_g}^2 & {\sigma_g}^2 & \dots & {\sigma_g}^2 \\ {\sigma_g}^2 & {\sigma_2}^2 + {\sigma_g}^2 & \dots & {\sigma_g}^2 \\ & \dots & & \\ {\sigma_g}^2 & {\sigma_g}^2 & \dots & {\sigma_n}^2 + {\sigma_g}^2 \end{pmatrix}$$

praktisches Beispiel: Konstruktion einer Kovarianzmatrix

6 Studenten in 3 Gruppen messen jeweils mit eigenem Messgerät vom gleichen Typ pro Gruppe, es gibt eine von allen angewandte „Theorie-Korrektur“ mit Unsicherheit.

Fehlerbeiträge:

- Systematischer Fehler eines Messgeräts: Δ_s (korreliert innerhalb einer Gruppe, d.h. Studierende 1-2, 3-4 und 5-6, unabhängig zwischen den Gruppen)
- Theoriefehler: Δ_t (korreliert für allen Messungen)
- einzelne unabhängige Messunsicherheiten: $\Delta_{f1}, \ldots, \Delta_{f6}$

Jede Messung hat die Gesamtunsicherheit

$$\Delta g_i = \sqrt{{\Delta_{f_i}}^2 + {\Delta_s}^2 + {\Delta_t}^2}$$

Konstruktionsprinzip:

- Quadrate der *Gesamt-unsicherheiten* stehen in der Diagonalen
- *gemeinsame Unsicherheiten* werden zu den betreffenden Nebendiagonalelementen quadratisch addiert

Anm: unabhängige Unsicher-heiten tauchen nur in der Diagonalen auf

$$V = \begin{pmatrix}
{\Delta g_1}^2 & \Delta_s^2 + \Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 \\
\Delta_s^2 + \Delta_t^2 & {\Delta g_2}^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 \\
\Delta_t^2 & \Delta_t^2 & {\Delta g_3}^2 & \Delta_s^2 + t^2 & \Delta_t^2 & \Delta_t^2 \\
\Delta_t^2 & \Delta_t^2 & \Delta_s^2 + \Delta_t^2 & {\Delta g_4}^2 & \Delta_t^2 & \Delta_t^2 \\
\Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 & {\Delta g_5}^2 & \Delta_s^2 + \Delta_t^2 \\
\Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_t^2 & \Delta_s^2 + \Delta_t^2 & {\Delta g_6}^2
\end{pmatrix}$$

Implementierung in PhyPraKit

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

```
def 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
  """

  # construct a numpy array with diagonal elements
  su = np.array(sig)
  V = np.diagflat(su*su)

  # if given, add off-diagonal components
  if sigc != []:
    for s in sigc:
      sc = np.array(s)
      V += np.outer(sc, sc)
  return V
```

Nachtrag: Fehlerfortpflanzung mit Kovarianzen

Für nicht verschwindende Kovarianzen $V_{ij} = \mathrm{cov}(x_i, x_j)$ lautet das
linearisierte Fehlerfortpflanzungsgesetz

$$y(x_1, \ldots, x_n) \Rightarrow \sigma_y^2 \simeq \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\frac{\partial y}{\partial x_i} \right) V_{i,j} \left(\frac{\partial y}{\partial x_j} \right)$$

Spezialfälle:

$$y = x_1 \pm x_2$$

$$\Rightarrow {\sigma_y}^2 = {\sigma_1}^2 + {\sigma_2}^2 \pm 2\,\mathrm{cov}(x, y)$$

$$y = x_1 \cdot x_2 \text{ oder } y = \frac{x_1}{x_2}$$

$$\Rightarrow \left(\frac{\sigma_y}{y} \right)^2 \simeq \left(\frac{\sigma_1}{x_1} \right)^2 + \left(\frac{\sigma_2}{x_2} \right)^2 \pm 2 \frac{\mathrm{cov}(x, y)}{xy}$$

Beispiel: Fehlerfortpflanzung mit Korrelation

Beispiel:

$$y = x_1 - x_2$$

$$\mu_1 = \mu_2 = 10,\ \sigma_1 = \sigma_2 = 1,\ \rho = 0$$

$$\Rightarrow\ \sigma_y = 1,4$$

aber mit Korrelationskoeffizient **ρ=1**

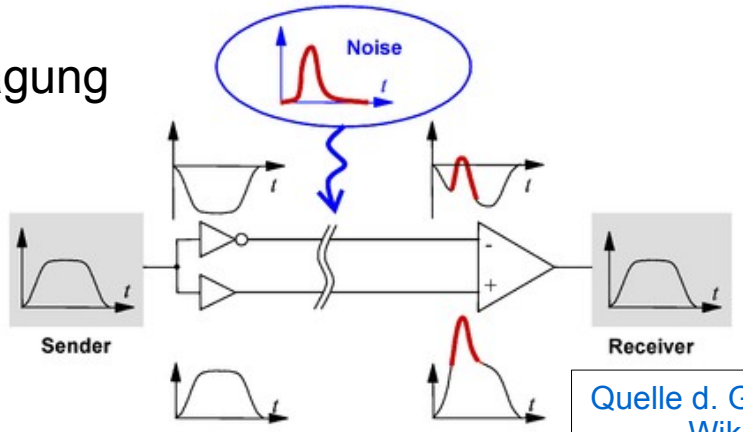
$$\Rightarrow\ \underline{\sigma_y = 0}\ !$$

Technische Anwendung:

„Differentielle“ Signalübertragung

Der Trick:

- gemeinsame Übertragung von Signal und invertiertem Signal
- zufällige Störungen betreffen beide gleichermaßen
- Differenzbildung beim Empfänger eliminiert Störungen



Quelle d. Grafik: Wikipedia

Also: Bitte nie die Kovarianzen vergessen !

Jupyter-Tutorial „Fehlerrechnung“

Mehr zur Konstruktion der Kovrianzmatrix finden Sie im jupyter-Tutorial

Fehlerrechnung.ipynb

http://etp.kit.edu/~quast/jupyter/jupyterTutorial.html

→ Fehlerrechnung.ipynb , Kap. 2.4