Least-Square-Fit mit Python (Anpassung) Stand: 14.5.12

Nachdem Python (unter Linux >ipython -pylab, windows: python(x,y)**) gestartet ist, muss zunächst das SCIPY-OPTIMIZE-Modul geladen werden**
(Falls es Scipy nicht gibt muss es installiert werden, Linux: Package suchen/installieren):
Hilfe: entweder: help('Begriff') oder ? Begriff

> import scipy.optimize
>? scipy.optimize
>? scipy.optimize.leastsq
Description:

Return the point which minimizes the sum of squares of M (non-linear) equations in N unknowns given a starting estimate, x0, using a modification of the Levenberg-Marquardt algorithm.

*x = arg min(**sum(func(y)****2,axis=0))*

Definiere eine Gerade-Funktion:

>line = lambda x,m,b: x*m+b

Lade die Daten line.dat ein:
d=loadtxt('line.dat')

Plote die Daten:
>plot(d[:,0],d[:,1],'o')

So, nun zurAnpssung: Sie benötigt eine Funktion, die zu minimieren ist:
>errfunc= lambda p,x,y: (y-line(x,p[0],p[1]).
Wir nennen Sie errfunc, sie enthält die Fehler/Abweichungen zwischen Daten y und Modellwerten line(x,p). Die Parameter der Gerade sind in einem Vektor abzubilden: p[0]= Steigung, p[1]=Achsenabschnitt

Die Funktion leastsq minimiert die Quadratischen Abweichungen, dazu muss die Funktion mit den Abweichungen an die Funktion leastsq übergeben werden.
Der Aufruf der Fit-Routine sieht dann folgendermaßen aus:
>p, success = scipy.optimize.leastsq(errfunc, [0, 0], args=(d[:,0],d[:,1]))
[**0, 0] ist der Startwerte-Vektor, er muss möglichst gute Startwerte enthalten, d.h. die Modellfunktion(Startwerten) sollte in der Nähe der Daten liegen (eventuell Plot).**
args sind die x- und die y werte die die Funktion errfunc zusätzlich zu dem Startwerten für p benötigt.
Ergebnis:
>success
Wenn sucess= 1 alles ok. Aufruf von p zeigt das Fitergebniss an:
>p
Für Rückgabe mit Fehlern etc muss der Parameter output auf 'True' gesetzt werden:
>p, covar, info, mess, ierr=scipy.optimize.leastsq(errfunc, [0, 0],args=(d[:,0],d[:,1]),full_output=True)
Fitergebnis
>p array([7.62396107, 1.96773393])
Meldung der Fit-Routine:
>mess: Both actual and predicted relative reductions in the sum of squares\n are at most 0.000000'
Klingt gut.

Covar ist die Kovarianzmatrix des Fits. Sie sagt etwas darüber aus, ob die Fit-Parameter linear abhängig sind (Nicht-Diagonal-Elemente sollten näherungsweise Null sein!

>Covar:
array([[0.00018762, -0.00365854],
[-0.00365854, 0.09634148]])

Die Diagonalelemente sind die Varianzen der angepassten Parameter, d.h. Wurzel daraus sind die Standardabweichungen:

>dm=sqrt(covar[0,0])=0.013697
>db=sqrt(covar[1,1])=0.310

Umrechnen in relative Fehler (in %):

db/p[1]*100=15.773943605303131
Steigung 15.7% Fehler
dm/p[0]*100=0.17966181176251236
Achsenabschnitt 0.17%

<u>Vergleich mit Gnuplot:</u>

>line(x,m,b)=m*x+b
>fit line(x,m,b) "line.dat" via m,b
Ergebnis:

Final set of parameters	Asymptotic Standard Error
m = 7.62397	+/- 0.4871 (6.389%)
b = 1.9674	+/- 11.04 (561.1%)

Da sagt etwas über die Bewertung der Fitfehler aus! Die verschiedenen Programme (Python/Gnuplot) kommen zu unterschiedlichen Bewertungen!

<u>Gaussfit unter Python (kurz, sollte jetzt klar sein):</u>

>g=loadtxt('gauss.dat')
>plot(g[:,0],g[:,1],'o')
>gauss = lambda x,m,s,yo: yo*exp(-(x-m)*(x-m)/2/s/s)
>errfunc = lambda p, x, y: gauss(x,p[0],p[1],p[2]) – y

Gute Startwerte [2000, 50, 40] sind wichtig! Teste:

>plot(g[:,0],gauss(g[:,1],2000,50,40),')
>p, covar, info, mess, ierr = scipy.optimize.leastsq(errfunc, [2000, 50, 40], args=(g[:,0],g[:,1]), full_output=True)
>p: array([2000.21950082, 79.02805859, 34.65196189])

Plote Fit-Ergebnis:

>plot(g[:,0],gauss(g[:,0],p[0],p[1],p[2]))
>covar
array([[5.28212572e-01, 9.11264956e-08, 2.19727825e-08],
[9.11264956e-08, 5.28212269e-01, -1.15803930e-01],
[2.19727825e-08, -1.15803930e-01, 7.61657503e-02]])

Berechne relative Fehler:

>errm= sqrt(covar[0,0])
>errm/p[0]*100=0.036335129283903006
>errs= sqrt(covar[1,1])
>errs/p[1]*100=0.036335129283903006
>erryo= sqrt(covar[2,2])
>erryo/p[2]*100=0.79643811207973902