Kapitel 4

Die Methode der kleinsten Quadrate

Die Methode der kleinsten Quadrate wird benützt, um die unbekannten Parameter einer Theoriefunktion aus gemessenen Daten zu bestimmen. Wir haben sie schon im Kap. 1.3.5 kennengelernt und angewandt auf einfache Probleme mit nur einem Parameter. In diesem Kapitel werden wir Theoriefunktionen mit mehreren Parametern betrachten und dabei zwischen linearen (Kap. 4.1) und nicht-linearen (Kap. 4.2) Funktionen unterscheiden. Die Methode ist dort anwendbar, wo die Fehler normalverteilt sind (siehe Kap. 3.2). Ist dies nicht der Fall, sollte man die Maximum-Likelihood-Methode (Kap. 5) benützen.

4.1 Anpassung einer linearen Funktion

4.1.1 Allgemeine Theorie; Fehler der Parameter

Die Messdaten seien

$$x_i\,, y_i \pm \sigma_i\,, i = 1 \cdots n$$

Die Fitfunktion (Theoriefunktion) ist linear in den Parametern $a_1 \cdots a_m$

$$\boxed{f(x;\vec{a}) = \sum_{j=1}^{m} a_j\, h_j(x)} \tag{4.1}$$

wobei $\vec{a} \equiv (a_1, \cdots, a_m)$ der m-dimensionale Parameter-Vektor ist. Die Definition von χ^2 ist (Kap. 2.4)

$$\boxed{\chi^2(\vec{a}) \equiv \sum_{i=1}^{n} \frac{[y_i - f(x_i;\vec{a})]^2}{\sigma_i^2}} \tag{4.2}$$

Die optimale Schätzung für die Parameter $a_1 \cdots a_m$ erhalten wir aus der Forderung, dass χ^2 minimal sei. Dies bedeutet, dass die partiellen Ableitungen $\partial\chi^2/\partial a_k$ verschwinden. Damit erhalten wir ein System von m gekoppelten linearen Gleichungen ($k = 1 \cdots m$)

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{n} g_i\, [y_i - f(x_i;\vec{a})] \frac{\partial f(x_i;\vec{a})}{\partial a_k} = 0 \tag{4.3}$$

oder mit Gl. 4.1

$$\sum_{i=1}^{n} g_i\, y_i\, h_k(x_i) = \sum_{j=1}^{m} \left[a_j \sum_{i=1}^{n} g_i\, h_j(x_i) h_k(x_i) \right] \tag{4.4}$$

wobei $g_i \equiv 1/\sigma_i^2$ die Gewichte sind.

Dieses Gleichungs-System kann als eine Matrizen-Gleichung geschrieben werden

$$\vec{b} = S\,\vec{a} \quad \text{oder in Komponenten} \quad b_k = \sum_{j=1}^{m} S_{kj} a_j\,, \quad k = 1 \cdots m \tag{4.5}$$

Die Komponenten des m-dimensionalen Vektors $\vec{b}$ und der $m \times m$-Matrix S folgen aus Gl. 4.4

$$\boxed{b_k = \sum_{i=1}^{n} g_i\, y_i\, h_k(x_i)} \quad \text{und} \quad \boxed{S_{kj} = S_{jk} = \sum_{i=1}^{n} g_i\, h_j(x_i)\, h_k(x_i)} \tag{4.6}$$

Die Lösung erfolgt durch Matrizen-Inversion

$$\boxed{\vec{a} = S^{-1}\,\vec{b} = C\,\vec{b}} \tag{4.7}$$

Fehler der Parameter
$C = S^{-1}$ ist die **Kovarianz-Matrix** (Kap. 3.2.3), wie folgende Betrachtung deutlich macht. Die Fehler der Parametern a_j folgen aus dem Fehlerfortpflanzungsgesetz

$$\sigma_{a_j}^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i} \right)^2$$

Die Abhängigkeit der Parametern a_j von den Messdaten y_i wird sofort ersichtlich, wenn wir Gl. 4.7 in Komponenten schreiben

$$a_j = \sum_{k=1}^{m} C_{jk} b_k = \sum_{k=1}^{m} \left[C_{jk} \sum_{i=1}^{n} g_i\, y_i\, h_k(x_i) \right]$$

Damit erhalten wir

$$\sigma^2_{a_j} = \sum_{i=1}^{n} \sigma_i^2 \left(\sum_{k=1}^{m} C_{jk} g_i \, h_k(x_i) \right)^2 = \sum_{i=1}^{n} g_i \left(\sum_{k=1}^{m} \sum_{l=1}^{m} C_{jk} h_k(x_i) \, C_{jl} h_l(x_i) \right)$$
$$= \sum_{k=1}^{m} \sum_{l=1}^{m} \left(C_{jk} C_{jl} \sum_{i=1}^{n} g_i h_k(x_i) \, h_l(x_i) \right) = \sum_{k=1}^{m} \sum_{l=1}^{m} C_{jk} C_{jl} S_{kl}$$

Mit $C\,S = I$ oder in Komponenten $\sum_{k=1}^{m} C_{jk} S_{kl} = I_{jl} = \delta_{jl}$ folgt daraus

$$\sigma^2_{a_j} = \sum_{l=1}^{m} \delta_{jl} C_{jl} = C_{jj} \tag{4.8}$$

Die zweite Ableitung von $\chi^2(\vec{a})$ (Gln. 4.1 und 4.3) ist

$$\frac{\partial^2 \chi^2(\vec{a})}{\partial a_j \partial a_k} = 2 \sum_{i=1}^{n} g_i \, h_j(x_i) \, h_k(x_i) = 2 S_{jk} \tag{4.9}$$

Im Folgenden betrachten wir nur einen Parameter a oder mehrere *unkorrelierte* Parameter. Für einen Parameter a ist damit die zweite Ableitung umgekehrt proportional zum Quadrat des Fehlers

$$\frac{d^2\chi^2(a)}{da^2} = 2S = \frac{2}{C} = \frac{2}{\sigma_a^2} \tag{4.10}$$

In der Nähe des Minimums können wir eine Taylor-Entwicklung machen

$$\chi^2(a) = \chi^2(a_0) + (a - a_0) \left(\frac{d\chi^2(a)}{da} \right)_{a=a0} + \frac{(a-a_0)^2}{2} \left(\frac{d^2\chi^2(a)}{da^2} \right)_{a=a0}$$

Die erste Ableitung ist Null im Minimum ($\chi^2(a_0) \equiv \chi^2_{min}$) und mit Gl. 4.10 erhält man

$$\boxed{\chi^2(a) = \chi^2_{min} + \frac{(a-a_0)^2}{\sigma_a^2}} \tag{4.11}$$

Dies ist ein **Parabel** mit den folgenden wichtigen Funktionswerten

$$\chi^2(a_0) \equiv \chi^2_{min} \qquad \boxed{\chi^2(a_0 \pm \sigma_a) = \chi^2_{min} + 1} \qquad \chi^2(a_0 \pm 2\sigma_a) = \chi^2_{min} + 4 \tag{4.12}$$

Diese nützlichen Relationen gelten auch für nicht-lineare Funktionen, da man für kleine Fehler σ_a die Funktion mit eine Taylor-Entwicklung linearisieren kann (siehe Kap. 4.2.2). Ein Beispiel mit einem Parameter haben wir schon im Kap. 1.3.5 gegeben. Im Kap. 4.1.2 und 4.2.1 werden Beispiele mit zwei unkorrelierten Variablen besprochen. Für korrelierte Variable werden diese einfachen Relationen durch die Kovarianzen komplizierter. Beispiele für zwei korrelierte Variable sind in den Kap. 3.2.2 und 4.2.1 gegeben.

4.1.2 Anpassung einer Geraden

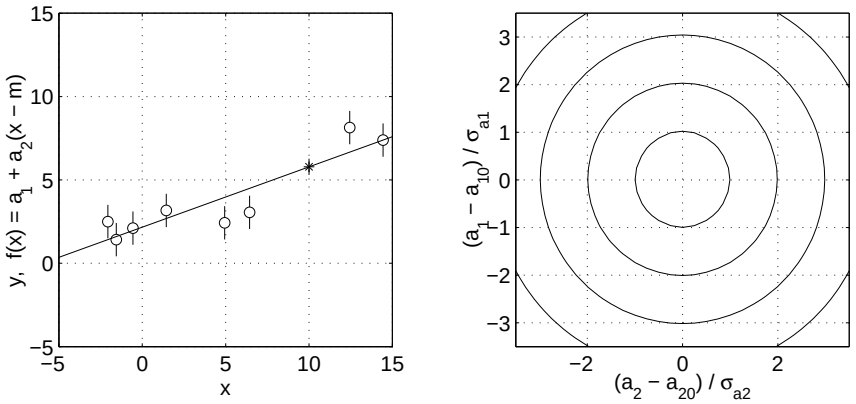

Abbildung 4.1: Anpassung einer linearen Funktion mit der Methode der kleinsten Quadrate. Links Monte-Carlo-Daten (x_i, $y_i \pm \sigma_i$) und Fitfunktion $f(x) = a_1 + a_2(x - \langle x \rangle)$. Der Punkt bei $x = 10$ ist ein interpolierter Wert mit seinem Fehler. Rechts: Höheliniendarstellung der $\chi^2(a_1, a_2)$-Funktion (1-, 2-, 3- und 4-σ Kontouren).

Ein einfaches Beispiel ist die Anpassung einer Geraden an die Messdaten. Die Fitfunktion ist (Gl. 4.1 mit $\vec{a} = (a_1, a_2)$, $h_1(x) = 1$ und $h_2(x) = x$)

$$f(x) = a_1 + a_2 \, x \tag{4.13}$$

Der Vektor $\vec{b}$ und die Matrix S sind (Gl. 4.6, $\sum \equiv \sum_{i=1}^{n}$)

$$\vec{b} = \begin{pmatrix} \sum g_i \, y_i \\ \sum g_i \, y_i \, x_i \end{pmatrix} \qquad S = \begin{pmatrix} \sum g_i & \sum g_i \, x_i \\ \sum g_i \, x_i & \sum g_i \, x_i^2 \end{pmatrix} \tag{4.14}$$

Die Kovarianz-Matrix ist

$$C = \begin{pmatrix} \sigma^2_{a1} & \mathrm{cov}(a_1, a_2) \\ \mathrm{cov}(a_1, a_2) & \sigma^2_{a2} \end{pmatrix} = S^{-1} = \frac{1}{\Delta} \begin{pmatrix} \sum g_i \, x_i^2 & -\sum g_i \, x_i \\ -\sum g_i \, x_i & \sum g_i \end{pmatrix} \tag{4.15}$$

wobei Δ die Determinante der Matrix S ist

$$\Delta = \left(\sum g_i \right) \left(\sum g_i \, x_i^2 \right) - \left(\sum g_i \, x_i \right)^2 \tag{4.16}$$

Mit $\vec{a} = C\vec{b}$ erhält man

$$a_1 = \frac{1}{\Delta}\left[\left(\sum g_i x_i^2\right)\left(\sum g_i y_i\right) - \left(\sum g_i x_i\right)\left(\sum g_i x_i y_i\right)\right]$$
$$a_2 = \frac{1}{\Delta}\left[\left(\sum g_i\right)\left(\sum g_i x_i y_i\right) - \left(\sum g_i x_i\right)\left(\sum g_i y_i\right)\right]$$

```
%-------------------------------------------------------------------------
%    MATLAB program 4.1: linear fit (correlated and uncorrelated)
%    ------------------------------------------------------------
load linearData;
x = linearData(:,1);
y = linearData(:,2);                        sig = linearData(:,3);
m = sum(x./sig.^2)/sum(1 ./sig.^2);
for i = 1:2
   fprintf('\nLinear fit: f(x) = a1 + ')
   if i == 1
      xm = 0;                               fprintf('a2*x\n')
   else
      xm = m;                               fprintf('a2*(x - m)\n');
   end
   data = [x-xm y sig];
   [a C] = linefit(data);                   sig1 = sqrt(C(1,1));
   sig2 = sqrt(C(2,2));                     rho = C(1,2)/(sig1*sig2);
   fprintf('   a1 = %5.3f +- %5.3f',a(1),sig1);
   fprintf('   a2 = %5.3f +- %5.3f',a(2),sig2);
   fprintf('  rho = %5.3f',rho);
   x0 = 10;                                 fx0 =  a(1) + a(2)*(x0-xm);
   d = [1; x0-xm];                         sigf = sqrt(d'*C*d);
   simple = sqrt(sig1^2 + sig2^2*(x0-xm)^2);
   fprintf('\n   X0 = %4.2f,  f(x0) = %4.2f +-  %4.2f',x0,fx0,sigf)
   fprintf(' (%4.2f)\n',simple)
end

%-------------------------------------------------------------------------
% Linear fit: f(x) = a1 + a2*x
%    a1 = 2.167 +- 0.442   a2 = 0.362 +- 0.060  rho = -0.600
%    X0 = 10.00,  f(x0) = 5.79 +-  0.48 (0.74)
%
% Linear fit: f(x) = a1 + a2*(x - m)
%    a1 = 3.775 +- 0.354   a2 = 0.362 +- 0.060  rho = -0.000
%    X0 = 10.00,  f(x0) = 5.79 +-  0.48 (0.48)
```



```
%-------------------------------------------------------------------------
function [a, C] = linefit(data)
%------------------------------
x = data(:,1);                              y = data(:,2);
g = 1 ./ data(:,3).^2;                      Sg = sum(g);
Sgx = sum(g.*x);                            Sgx2 = sum(g.*x.^2);
Det = Sg*Sgx2 - Sgx^2;                      b = [sum(g.*y); sum(g.*y.*x)];
C = [Sgx2, -Sgx; -Sgx, Sg]/Det;             a = C*b;
%-------------------------------------------------------------------------
```

Ein Beispiel haben wir schon im Kap. 3.2.2 besprochen. Dabei haben wir die optimalen Parameter, ihre Fehler und Korrelation grafisch ermittelt (Abb. 3.8). Die analytische Lösung gibt das gleiche Resultat (siehe Programm 4.1). Der Fit wird mit der Funktion `[a, C] = linefit(data)` durchgeführt. Im Programm 4.1 wird sie zweimal aufgerufen (`for i = 1:2`), einmal mit den unveränderten Daten und einmal mit modifizierten x-Werten, $x' = x - \bar{x}$ (`data = [x-xm y sig]`), wobei $\bar{x} = \sum g_i x_i / \sum g_i$ der mit den Fehlern von y gewichtete Mittelwert von x ist. Damit der Kovarianz (Gl. 4.15)

$$\mathrm{cov}(a1,\, a2) \propto \sum g_i\, x_i' = \sum g_i\, x_i - \bar{x} \sum g_i = 0$$

Mit den Fitparametern kann man zu jedem x-Wert den Funktionswert berechnen (Inter- oder Extrapolation). Dessen Fehler wird nur im Falle $\rho = \mathrm{cov}(a1,\, a2)/(\sigma_1\sigma_2) = 0$ durch das einfache Fehlerfortpflanzungsgesetz (`simple = sqrt(sig1^2 + sig2^2*(x0-xm)^2)`) gegeben. Für $\rho \neq 0$ sollte man Gl. 3.23 oder in der Matrizenschreibweise Gl. 3.25 benützen. Die Matrizenform des Fehlerfortpflanzungsgesetzes kann man direkt in MATLAB programmieren: `d = [1; x0-xm]; sigf = sqrt(d'*C*d)`. Das Resultat der Interpolation (Abb. 4.1, Output des Programms 4.1) ist, wenn man das richtige Fehlerfortpflanzungsgesetz benützt, für beide Fälle gleich.

4.1.3 Anpassung eines Polynoms

Die Anpassung eines Polynoms an gemessene Daten ist die direkte Verallgemeinerung der Anpassung einer Geraden. Die Fitfunktion ist (Gl. 4.1 mit $h_j(x) = x^{j-1}$)

$$f(x;\, \vec{a}) = \sum_{j=1}^{m} a_j x^{j-1} \tag{4.17}$$

Die optimalen Schätzwerte der Parameter $a_1 \cdots a_m$ erhalten wir aus den Matrizen-Gleichungen 4.6 und 4.7. Diese Gleichungen sind in der MATLAB-Funktion `[a, C] = polfit(data,m)` (Programm 4.2) programmiert, wobei `a` der m-dimensionale Parametervektor (Spaltenvektor) ist und `C` die $(m \times m)$-Kovarianzmatrix. Die Potenzen von x werden für alle n x-Werte in der $(n \times m)$-Matrix `hx` gespeichert. Die n Funktionswerte werden mit `hx*a` berechnet. Wird die Funktion `polfit(data,m)` mit einer $n \times 2$-Matrix `data` (Spalte 1: x-Werte; Spalte 2: y-Werte) aufgerufen, so werden die y-Fehler alle gleich Eins gesetzt. Damit ist der χ^2-Wert des Minimums χ_0^2 willkürlich. Deshalb wird die Kovarianzmatrix mit dem Faktor $\chi_0^2/(n-m)$ multipliziert und $\chi_0^2 = n - m$ gesetzt ($n - m$ ist die Zahl der Freiheitsgrade). Dieses Verfahren gibt die richtigen Fehler und Kovarianzen der Parameter, wenn die y-Fehler alle gleich gross sind (die absoluten Werte aber unbekannt sind). Ein unabhängiger χ^2-Test, wie gut die Fitfunktion mit den Daten übereinstimmt, haben wir in diesem Fall nicht mehr.

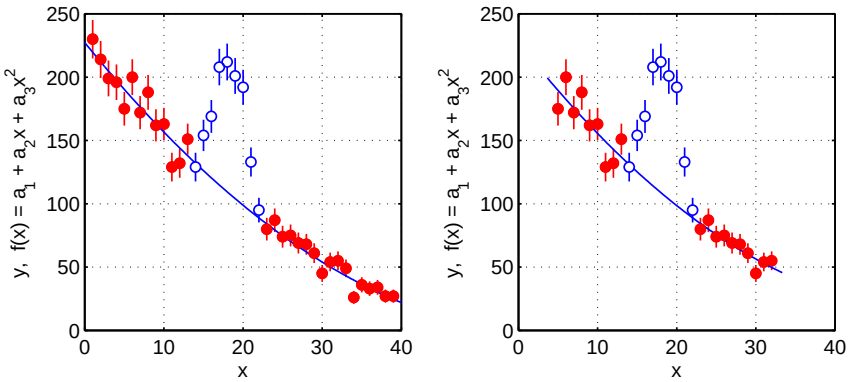

Abbildung 4.2: Gauss-Peak mit quadratischem Untergrund (Monte-Carlo-Daten). Links: Fit des Untergrundes in einem grossen Bereich; Rechts: in einem kleinen Bereich

Ein Beispiel, die Anpassung eines quadratischen Untergrundes unter einem Gauss-Peak, ist in Abb. 4.2 (Programm 4.2) gegeben. Die Monte-Carlo-Daten wurden mit den Parametern $a_1 = 220$, $a_2 = -7.4$ und $a_3 = 0.063$ generiert (Programm 3.4). Die Schätzwerte sind damit in Übereinstimmung, unabhängig vom gewählten Untergrundbereich (siehe die Resultate von Programm 4.2).

Die Fläche N des Gauss-Peaks ist ($x_a = k_a - 0.5$, $x_b = k_b + 0.5$)

$$N = T - B, \quad T = \sum_{k=k_a}^{k_b} y_k$$
$$B = \int_{x_a}^{x_b} f(x)\,dx = a_1(x_b - x_a) + a_2(x_b^2 - x_a^2)/2 + a_3(x_b^3 - x_a^3)/3$$

Der Fehler ist (T und B sind unkorreliert)

$$\sigma_N = \sqrt{\sigma_T^2 + \sigma_B^2} = \sqrt{T + \sigma_B^2}$$

Da die Parameter a_1, a_2 und a_3 stark korreliert sind, müssen wir zur Berechnung des Fehlers σ_B das allgemeine Fehlerfortplanzungsgesetz (Gl. 3.25) benützen

$$\sigma_B^2 = \vec{d}^T\, C\, \vec{d}, \quad \vec{d}^T = (x_b - x_a,\ (x_b^2 - x_a^2)/2,\ (x_b^3 - x_a^3)/3)$$

Das Resultat ist (Programm 4.2) $N = 535 \pm 57$ in Übereinstimmung mit dem Wert $a_4 = 573$ des Monte-Carlo-Programms 3.4.

```
%-------------------------------------------------------------------------
%    MATLAB program 4.2: polynomial fit of background; peak area
%    -----------------------------------------------------------
load gb_data;
xa  = 13.5;                                     xb = 22.5;
% xa= 12.5;                                     xb = 24.5;
x   = gb_data(:,1);                             y = gb_data(:,2);
% x = gb_data(5:32,1);                          y = gb_data(5:32,2);
sig = sqrt(y);                                  G =  x >= xa &  x <= xb;
[a, C] = polfit([x(~G) y(~G) sig(~G)],3);       grid
plot_data([x(G) y(G) sig(G)],'b')
plot(x(~G),y(~G),'r.','MarkerSize',20)
ylabel('y,  f(x) = a_1 + a_2x + a_3x^2'),       xlabel('x')
d = [(xb - xa); (xb^2 - xa^2)/2; (xb^3 - xa^3)/3];
T = sum(y(G));                                  B = a'*d;
N = T - B,                                      sigN = sqrt(T + d'*C*d),

%-------------------------------------------------------------------------
%    xa     xb       a(1)       a(2)        a(3)        N
%   13.5   22.5     227(7)    -7.7(7)    0.065(15)    505(46)
%   12.5   24.5     227(7)    -7.8(7)    0.068(16)    535(57)
%   12.5   24.5     231(15)   -9(2)      0.09(5)      561(78)     range 5:32
```

```
%-------------------------------------------------------------------------
function [a, C] = polfit(data,m)
%------------------------------
%  [a, C] = polfit(data,m): (weighted) least squares fit to a polinomial
%  input:  array data with 2 or 3 columns: x(i), y(i),
%          (sig(i) optional), m: number of parameters
%  output:  a: parameters, C covariance matrix; plot and print of results
x = data(:,1);                          y = data(:,2);
n =length(x);                           sd = size(data);
if sd(2) > 2
   sig = data(:,3);                     % third column are errors
else
   sig = ones(n,1);                     % equal errors calculated
end
g = 1 ./(sig.*sig);                     % weights
hx = zeros(n,m);
for j = 1:m                             % powers of x
    hx(:,j) = x.^(j-1);
end;
S = zeros(m,m);                         b = zeros(m,1);
for k = 1:m
    for j = 1:m                         % calculate matrix
        S(k,j) = sum(g.*hx(:,k).*hx(:,j));
    end;
    b(k,1) = sum(g.*hx(:,k).*y);        % calculate vector
end;
C = inv(S);                             % covariance matrix
a = C*b;                                % parameters
fx = hx*a;                              % function values
chi2 = sum(g.*(y - fx).^2);             % chisquare
if sd(2) == 2                           % no errors in data points
   C = (chi2/(n-m))*C;                  % scale covariance matrix
   sig = sqrt(chi2/(n-m))*sig;          % and errors
   chi2 = n-m;
end
prob = Pchisqr(n-m,chi2);               % probability
for k = 1:m                             % calculate correlation matrix
    for j = 1:m
        rho(k,j) = C(k,j)/sqrt(C(k,k)*C(j,j));
    end;
end;
disp('least squares fit to polynomial a1 + a2*x + ... am*x^m-1')
disp('parameter +- error'),             format short e
disp([a sqrt(diag(C))]),                format short
disp('correlation matrix'),             disp(rho)
disp('chisquare, dof, probability'),    disp([chi2 n-m prob])
plot_data([x y sig]),                   dx = (max(x) - min(x))/100;
x1 = (min(x)-5*dx:dx:max(x)+5*dx)';     hx = zeros(length(x1),m);
for k = 1:m
    hx(:,k) = x1.^(k-1);                % powers of x
end;
plot(x1,hx*a,'g-')                      % function values
```


4.1.4 *Anpassung einer allgemeinen linearen Funktion

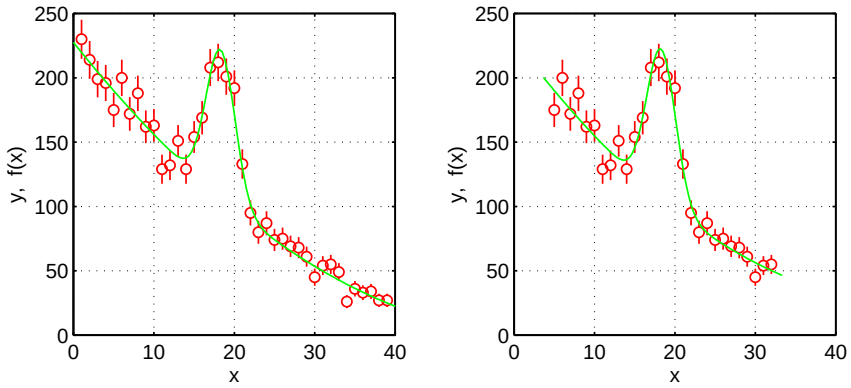

Abbildung 4.3: Gauss-Peak mit quadratischem Untergrund (Monte-Carlo-Daten). Links: Fit in einem grossen Bereich; Rechts: in einem kleinen Bereich

Um eine allgemeine lineare Funktion an gemessene Daten anzupassen, müssen wir die MATLAB-Funktion `[a, C] = polfit(data,m)` (Programm 4.2) abändern. Statt die Potenzen von x, müssen die Funktionswerte $h_k(x)$ (Gl. 4.1, $k = 1 \cdots m$) für alle n x-Werte in der $(n \times m)$-Matrix `hx` gespeichert werden. Dies wird in der MATLAB-Funktion `[a, C] = linfit(data,func)` (Programm 4.3) getan mit dem Befehl `hx = feval(func,x)`. `func` ist ein "String" mit dem Name der Funktion, die die Funktionswerte $h_k(x)$ berechnet.

Ein Beispiel, die Anpassung eines Gauss-Peaks mit quadratischem Untergrund an Monte-Carlo-Daten (Programm 3.4), ist im Abb. 4.3 (Programm 4.3) gegeben. Dabei

wurde angenommen, dass die Position ($\mu = 18.3$) und die Breite ($\sigma = 1.8$) des Gauss-Peaks bekannt seien. Neben den drei Parametern des quadratischen Untergrunds gibt es als vierten Parameter die Fläche des Gauss-Peaks. Die Funktionen $h_k(x)$ sind damit

$$h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = x^2 \quad \text{und} \quad h_4(x) = \frac{1}{\sqrt{2\pi}\,\sigma}\, e^{-(x-\mu)^2/2\sigma^2}$$

Sie werden mit der Funktion `f = GaussBack(x)` berechnet und der Fit wird mit dem Befehl `[a, C] = linfit([x y sig],'GaussBack')` durchgeführt. Das Resultat (siehe Output Programm 4.3) ist für die Parameter des Untergrundes gleich wie vorher (Programm 4.2). Da in diesem Fit die Form des Gauss-Peaks als bekannt vorausgesetzt ist, ist seine Fläche genauer bestimmt als vorher.

```
%-------------------------------------------------------------------------
%    MATLAB program 4.3: linear fit of quadratic background and gaussian
%    -----------------------------------------------------------------
load gb_data;
x = gb_data(:,1);                              y = gb_data(:,2);
% x = gb_data(5:32,1);                         y = gb_data(5:32,2);
sig = sqrt(y);
[a, C] = linfit([x y sig],'GaussBack');        grid
ylabel('y,  f(x)'),                            xlabel('x')

%-------------------------------------------------------------------------
%        a(1)       a(2)        a(3)        a(4)
%       227(7)    -7.8(7)    0.067(14)    518(40)
%       230(13)   -8.3(15)   0.08(4)      526(44)     range 5:32

%-------------------------------------------------------------------------
%  function [a, C] = linfit(data,func)
%  same as function [a, C] = polfit(data,m), but with the function
%  values calculated by hx = feval(func,x);

%-------------------------------------------------------------------------
function f = GaussBack(x)
%------------------------
%   f = GaussBack(x): Gaussian peak with quadratic background
%   returns 4 functions: f1 = 1; f2 = x; f3 = x^2; f4 = gaussian
[n m] = size(x);
if n > m
   x = x';
```



```
end
m = 18.3;                                      sig = 1.8;
f = [ones(size(x)); x;  x .* x; gauss(x,m,sig)]';
%-------------------------------------------------------------------------
```

4.1.5 *Fehler der unabhängigen Variable x

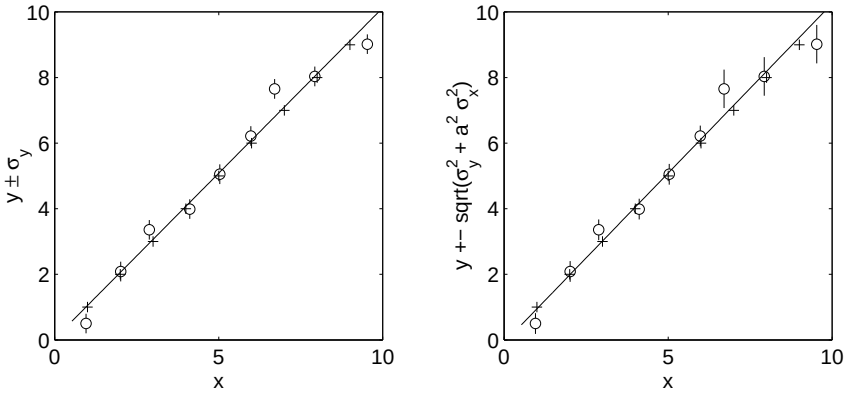

Abbildung 4.4: Anpassung einer Geraden $f(x) = ax + b$ an Monte-Carlo-Daten (+: exakte Punkte; o: mit der Auflösung "verschmiert"). Links: Nur die Fehler in y wurden berücksichtigt; Rechts: Fit mit Fehlern in x und y.

Bisher sind wir davon ausgegangen, dass der Fehler in den unabhängige Variable x verschwindend klein ist. Dies ist meistens auch der Fall, da an genau bekannter Stelle x einen Wert y mit dem Fehler σ_y gemessen wird. Ist dies nicht der Fall, müssen wir auch den Fehler σ_x berücksichtigen. An einem Beispiel, die Anpassung einer Geraden $f(x) = ax + b$ an Monte-Carlo-Daten ($x_i \pm \sigma_{xi}$, $y_i \pm \sigma_{yi}$), werden wir zeigen, wie man das iterativ machen kann (Programm 4.4, Abb. 4.4). Zuerst wird ein χ^2-Fit mit $\sigma_i = \sigma_{yi}$ durchgeführt

$$\chi^2 = \sum_{i=1}^{n} \frac{(y_i - ax_i - b)^2}{\sigma_i^2}$$

und vorläufige Schätzwerte a_0', b_0' bestimmt. Der Fehler in $f_i = a_0' x_i + b_0'$ folgt aus dem Fehlerfortpflanzungsgesetz: $\sigma_{fi}^2 = (a_0')^2 \sigma_{xi}^2$ (für fest vorgegebene Werte a_0' und b_0'). Der Fehler in der Differenz $y_i - f_i$ ist $\sigma_i^2 = \sigma_{yi}^2 + (a_0')^2 \sigma_{xi}^2$ und mit diesen Fehlern wird

der nächste χ^2-Fit durchgeführt und bessere Schätzwerte a_0, b_0 bestimmt. Meistens weichen diese verbesserten Schätzwerte nicht wesentlich von den ersten Schätzwerten ab (d.h. die Unterschiede sind deutlich kleiner als der Fehler von a und b) und damit ist die Iteration beendet. Der zweite Schritt ist vor allem notwendig um die richtigen Fehler der Parameter und χ^2-Wert zu erhalten. Sind sowohl die Fehler in x als auch in y konstant, so gibt es keinen Unterschied zwischen den Parameterwerten der ersten und zweiten Iteration. Sind die x-Fehler dagegen stark von x abhängig, wie in unserem Beispiel, kann es deutliche Unterschiede geben (siehe die Resultate von Programm 4.4).

```
%------------------------------------------------------------------------
%    MATLAB program 4.4: linear fit with y and x errors
%    -----------------------------------------------
a = 1.0;                                  b = 0;
x = 1:9;                                  sigx = 0.1*[1 1 1 1 1 1 5 5 5];
y = a*x + b;                              sigy = 0.3;
error = randn(2,length(x));               % normal error distribution
xe = sigx.*error(1,:);                    ye = sigy*error(2,:);
x1 = x + xe;                              y1 = y + ye;
subplot(121),                             % only y errors
sig = sigy*ones(1,length(y));
par1 = polfit([x1' y1' sig'],2),          plot(x,y,'b+'),
ylabel('y \pm \sigma_y'),                 xlabel('x')
axis('square'),                           axis([0 10 0 10])
subplot(122),                             % x and y errors
sig = sqrt(sigy^2 + par1(2)^2 .* sigx.^2);
par2 = polfit([x1' y1' sig'],2),          plot(x,y,'b+')
axis('square'),                           axis([0 10 0 10])
ylabel('y +- sqrt(\sigma_y^2 + a^2 \sigma_x^2)'),    xlabel('x')

%--------- only y errors ------------------------ x and y errors -------
%      a = 0.930 +- 0.037,                      a = 0.968 +- 0.051
%      b = 0.42  +- 0.021,                      b = 0.28  +- 0.24
%  chi^2 = 16.8  (1.9 %),       dof = 7,     chi^2 = 9.9  (19.3 %)
%------------------------------------------------------------------------
```


4.2 Anpassung einer nicht-linearen Funktion

4.2.1 χ^2-Fit einer Exponentialfunktion

Im Kap. 4.2.2 werden wir ein allgemeiner Formalismus für die Anpassung nicht-linearer Funktionen und Beispielen in den Kap. 4.2.3, 4.2.4 kennen lernen. In diesem Kapitel wollen wir zuerst ein Beispiel mit grafischen Methoden behandeln und dabei die Bedeutueng der Korrelation zweier Parameter hervorheben (siehe dazu auch die Kap. 3.2.2 und 4.1.2).

Die Anpasung einer Exponentialfunktion

$$f(x) \propto e^{a\,x}$$

ist ein wichtiges Beispiel einer Funktion mit einem nicht-linearen Parameter a. In diesem Abschnitt werden wir verschiedene χ^2-Fits vergleichen. In Kap. 4.2 werden wir eine Anpassung mit der Maximum-Likelihood-Methode durchführen.

Ein exponentielles Zeitspektrum (Histogramm) wurde mit einem TDC gemessen (siehe Kap. 2.3, Abb. 2.8). Die Messdaten sind

$$t_i,\, y_i \pm \sigma_i, \;\; i = 1 \cdots n, \;\; t_1 = \Delta t/2,\, t_n = T - \Delta t/2$$

Die Binbreite Δt ist konstant. y_i ist die Zahl der Ereignisse in Bin i, d.h. alle Ereignisse mit $t_i - \Delta t/2 \le t < t_i + \Delta t/2$. Die totale Zahl der Ereignisse ist $N = \sum_{i=1}^{n} y_i$. Der Fehler folgt aus der Poisson-Statistik (Kap. 2.2) $\sigma_i = \sqrt{y_i}$.

Eine mögliche Fitfunktion ist

$$f_1(t;\, N_1,\, \lambda) = N_1\, e^{-\lambda t} \equiv N_1\, h_1(t;\, \lambda) \tag{4.18}$$

wobei N_1 ein linearer und λ ein nicht-linearer Parameter ist. N_1 ist die Zählrate zur Zeit $t = 0$ und λ ist die Rate eines Poisson-Prozesses, z.B. die Zerfallsrate eines radioaktiven Kerns oder, wie in unserem Beispiel, eine zufällige Stopprate (siehe Kap. 2.3). $\tau = 1/\lambda$ ist die Lebensdauer, bzw. der mittlere Zeitabstand zwischen zwei zufälligen Ereignissen. Der Erwartungswert für die Zahl der Ereignisse in Bin i ist damit

$$f_{1i} \equiv \int_{ti-\Delta t/2}^{ti+\Delta t/2} f_1(t)dt \simeq f_1(t_i)\Delta t \tag{4.19}$$

und der Erwartungswert für die Gesamtzahl der Ereignisse

$$\sum_{i=1}^{n} f_{1i} = \int_0^T f_1(t)dt = \frac{N_1}{\lambda}\left(1 - e^{-\lambda T}\right) \tag{4.20}$$

χ^2 ist eine Funktion beider Parameter

$$\chi^2(N_1,\,\lambda) = \sum_{i=1}^{n} \frac{[y_i - f_{1i}(N_1,\,\lambda)]^2}{y_i} \tag{4.21}$$

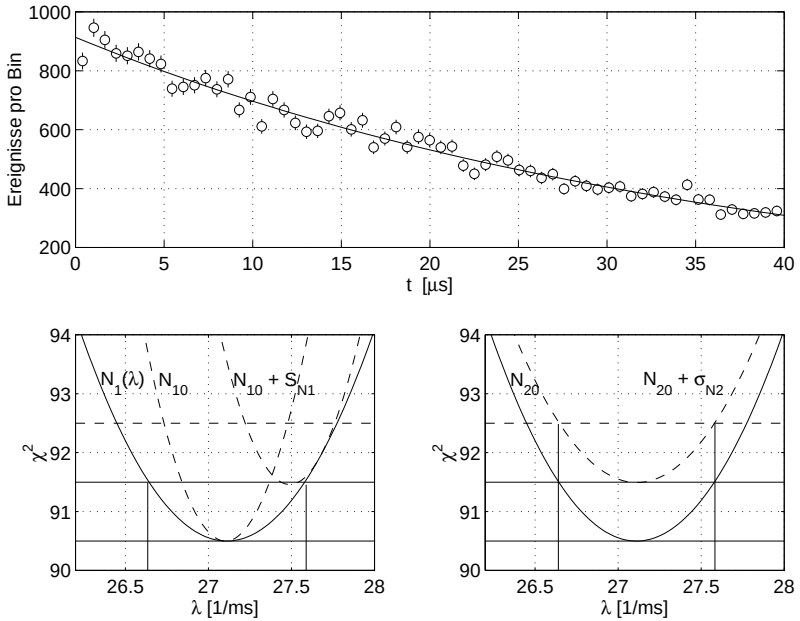

Abbildung 4.5: 1-Parameter χ^2-Fit einer Exponentialfunktion. Oben: Messdaten und Fitfunktion; Unten: χ^2 als Funktion von λ. Links: $f_1(t)$ mit $N_1(\lambda)$, N_{10} und $N_{10}+\sigma_{N1}$. Rechts: $f_2(t)$ mit N_{20} und $N_{20}+\sigma_{N2}$.

Die beste Schätzung für die Gesamtzahl der Ereignisse ist die gemessene Zahl der Ereignisse (Poisson-Statistik)

$$\sum_{i=1}^{n} f_{1i} = N = \sum_{i=1}^{n} y_i \tag{4.22}$$

und mit Gl. 4.20 folgt daraus, dass N_1 kein unabhängiger Parameter ist

$$N_1(\lambda) = \frac{\lambda\, N}{1 - e^{-\lambda T}} \tag{4.23}$$

χ^2 ist damit nur noch eine Funktion von λ und aus der graphische Darstellung dieser Funktion (Abb. 4.5) liest man den Schätzwert für λ und seiner Fehler ab (Gl. 4.12; siehe auch Kap. 1.3.5)

$$\lambda_0 = (27.11 \pm 0.47)\ \ \mathrm{ms}^{-1}$$

Man beachte, dass man nur mit N_1 als Funktion von λ (Gl. 4.23) die richtige χ^2-Funktion erhält. Für $N_1 = N_{10} \equiv N_1(\lambda_0)$ oder $N_1 = N_{10} + \sigma_{N1}$ erhält man zu kleine Fehler und bei nicht optimalem Schätzwert von N_1 sogar ein falscher Schätzwert für λ.

Eine "bessere" Fitfunktion ist

$$f_2(t;\, N_2,\, \lambda) = N_2\, \frac{\lambda e^{-\lambda t}}{1 - e^{-\lambda T}} \equiv N_2\, h_2(t;\, \lambda) \tag{4.24}$$

N_2 ist der Erwartungswert für die Gesamtzahl der Ereignisse

$$\sum_{i=1}^{n} f_{2i} = \int_0^T f_2(t)dt = N_2 \tag{4.25}$$

und ist unabhängig von λ. Deshalb ist f_2 eine "bessere" Fitfunktion als f_1. Der Schätzwert für N_2 ist (Poisson-Statistik)

$$N_{20} = N = \sum_{i=1}^{n} y_i \qquad \sigma_{N2} = \sqrt{N} \tag{4.26}$$

Für die Daten aus Abb. 4.5 ist das Resultat

$$N_{20} = 35521 \pm 188$$

und für λ erhalten wir den gleichen Wert (Abb. 4.5, Programm 4.5) wie vorher (Fitfunktion f_1). Jetzt ist das Resultat für λ jedoch unempfindlich für den genauen Wert von N_2. Das ist der praktische Vorteil mit unkorrelierten Parametern!

Der optimale 1-Parameter-Fit ist also ($N = \sum y_i$, $T = t_n + \Delta t/2$)

$$\boxed{\chi^2(\lambda) = \sum_{i=1}^{n} \frac{[y_i - N f(t_i;\, \lambda)\, \Delta t]^2}{y_i}} \quad \text{mit} \quad \boxed{f(t;\, \lambda) = \frac{\lambda \exp(-\lambda t)}{1 - \exp(-\lambda T)}} \tag{4.27}$$

Man beachte, dass die Theoriefunktion im experimentellen Bereich normiert ist!

```
%-----------------------------------------------------------------------
%    MATLAB program 4.5: 1 parameter fit of exponential
%    -----------------------------------------------
load exp_data1,                        data = exp_data1;
subplot(121)
plot_data(data),                       grid
axis([0 40 200 1000]),
xlabel('t  [\mus]'),                   ylabel('Ereignisse pro Bin')
x = data(:,1);                         y = data(:,2);
dt = x(2) - x(1);                      T = max(x) + dt/2;
l = (26.2:0.02:28.0);                  lambda = l*1.0e-3;
a = l(1);                              b = max(l);
chi2 = zeros(size(lambda));            N = sum(y);
for i = 1:length(lambda)
   fx = lambda(i)*exp(-lambda(i)*x)*dt/(1 - exp(-lambda(i)*T));
   chi2(i) = sum((y - N*fx).^2 ./ y);
end
subplot(122)
plot(l,chi2,'b-')
axis([a b 90 94]),                     hold on,
xlabel('\lambda [1/ms]'),              ylabel('\chi^2')
grid,                                  mchi2 = min(chi2);
plot([a b],[mchi2 mchi2],'r-'),        plot([a b],[mchi2+1 mchi2+1],'r-')
disp('specify 1 sigma points (2)')
[xs,ys] = ginput(2);                   lambda0 = (xs(2) + xs(1))/2;
plot([xs(1) xs(1)],[0 ys(1)],'k-'),    plot([xs(2) xs(2)],[0 ys(2)],'k-')
txt = sprintf(' = %5.2f +- %5.2f\n',lambda0,(xs(2) - xs(1))/2);
text(26.5,93.5,['\lambda_0 ' txt]);
subplot(121),
t = 0:45;                              lambda0 = lambda0*1.0e-3;
ft = lambda0*exp(-lambda0*t)*dt/(1 - exp(-lambda0*T));
plot(t,N*ft,'b-'),                       axis([0 40 200 1000])
%-----------------------------------------------------------------------
```

In beiden Fällen kann man auch einen 2-Parameter-Fit durchführen. Das Resultat für den linearen Parameter N_2 (bzw. N_1) kann man analytisch berechnen

$$\left(\frac{\partial\chi^2(N_2,\lambda)}{\partial N_2}\right)_{N2=N20} = 0 \quad \Rightarrow \sum_{i=1}^{n} \frac{y_i - f_{2i}}{y_i}\, h_{2i} = 0$$

oder

$$N_{20} = \frac{\sum_{i=1}^{n} h_{2i}}{\sum_{i=1}^{n} h_{2i}^2/y_i}$$

Dies ist nicht die gleiche Formel wie vorher (GL. 4.26) und auch das numerische Resultat ist unterschiedlich (die Differenz ist aber klein im Vergleich zum Fehler)

$$N_{20} = 35544 \pm 188$$

Warum? In der Formel für χ^2 (Gl. 4.21) wurde die Näherung $\sigma_i^2 = y_i$ gemacht, was für genügend grosse Werte von y_i (≥ 10) genügend genau ist. Der bessere Schätzwert ist aber $\sigma_i^2 = f_{2i}$ (bzw. f_{1i}), was aber zu praktischen Problemen führt, da die Fitfunktion von den zu schätzenden Parametern abhängt. Für kleine Werte von y_i sollte man die Maximum-Likelihood-Methode (Kap. 5) benützen. Im einfachen Fall (Fitfunktion 2) kann man den Schätzwert auch ohne Näherung bestimmen. Mit $\sigma_i^2 = f_{2i} = N_2 h_{2i}$ erhält man

$$\left(\frac{\partial\chi^2(N_2,\lambda)}{\partial N_2}\right)_{N2=N20} = 0 \quad \Rightarrow \sum_{i=1}^{n} \frac{y_i - f_{2i}}{N_{20} h_{2i}}\, h_{2i} = 0$$

und damit

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} f_{2i} = N_{20} \sum_{i=1}^{n} h_{2i} = N_{20}$$

```
%-----------------------------------------------------------------------
%    MATLAB program 4.6: contour plot of chi^2
%    ------------------------------------------
select = 2;  % 1: ft = N*exp(-lambda*t)
             % 2: ft = lambda*N*exp(-lambda*t)/(1 - exp(-lambda*T)
load exp_data1,                          xd = exp_data1(:,1);
yd = exp_data1(:,2);                     dt = xd(2) - xd(1);
weight = 1 ./ yd;                        T = max(xd) + dt/2;
sigma = -3:0.2:3;
lambda = 4.6855e-04*sigma + 2.7111e-02;   % [1/microsec]
if select == 1
   norm = 13.7*sigma + 1445.1;
else
   norm = 188*sigma + 35545;
end
[x,y] = meshgrid(lambda,norm);           chi2 = 0;
for i = 1:length(xd)
   if select ==1
       fx = y .* exp(-xd(i) .* x)*dt;
   else
       fx = y .* x .* exp(-xd(i) .* x)./(1-exp(-x*T))*dt;
   end
   chi2 = chi2 + weight(i)*(fx - yd(i)).^2;
```

```
end
v = (1:10).^2 + min(min(chi2))
contour(sigma,sigma,chi2,v,'k-')
hold on,                                    plot(0,0,'k+'),
grid,                                       axis('square')
xlabel('(\lambda - \lambda_0) / \sigma_\lambda'),
if select == 1
   ylabel('(N_1 - N_{10}) / \sigma_{N1}')
else
   ylabel('(N_2 - N_{20}) / \sigma_{N2}')
end
%-------------------------------------------------------------------------
```

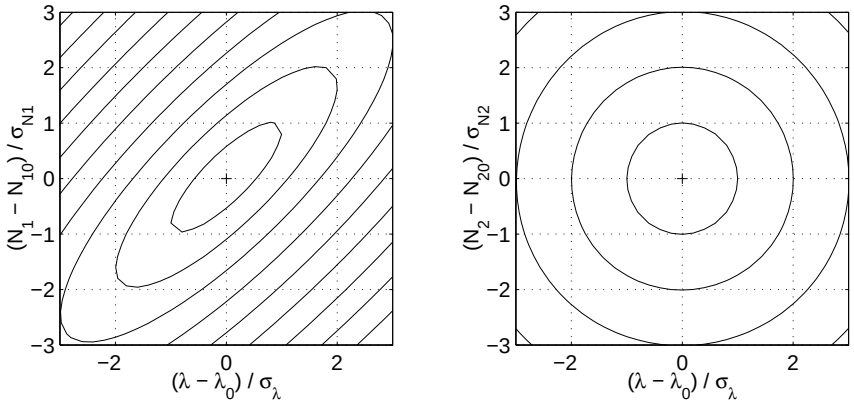

Abbildung 4.6: Kontour-Plot von $\chi^2(N,\lambda)$. Die 1-, 2-, 3-, $\cdots k$-σ Kovarianz-Ellipsen sind eingezeichnet. Die Achsen sind normiert: $(\lambda-\lambda_0)/\sigma_\lambda$ und $(N-N_0)/\sigma_N$. Links: Fitfunktion $f_1(t)$; Rechts: Fitfunktion $f_2(t)$.

Aus der grafischen Darstellung von χ^2 als Funktion zweier Parametern (λ und N_1, bzw. N_2; Abb. 4.6; Programm 4.6) ist die starke Korrelation zwischen λ und N_1 klar ersichtlich. Die Resultate für λ und N_2 sind oben schon gegeben. Für N_1 ist der Schätzwert

$$N_{10} = (1445.1 \pm 13.7)\ \ \mu\mathrm{s}^{-1}$$

Dies ist natürlich kein unabhängiges Resultat. Es lässt sich aus der Beziehung

$$N_1 = N_2 \frac{\lambda}{1-e^{-\lambda T}}$$

und mit dem Fehlerfortpflanzungsgesetz aus den Resultaten für λ und N_2 herleiten ($T = 40.53\ \mu$s).

4.2.2 Linearisierung der Fitfunktion

Eine Fitfunktion die nicht-linear von den Parametern $\vec{a} = (a_1 \cdots a_m)$ abhängt, kann man linearisieren durch eine Taylor-Entwicklung bis zur ersten Ordnung. Mit $\vec{a} = \vec{a}_0 + \delta\vec{a}$ erhält man

$$f(x;\vec{a}) = f(x;\vec{a}_0) + \sum_{j=1}^{m} \delta a_j \left(\frac{\partial f(x;\vec{a})}{\partial a_j}\right)_{\vec{a}=\vec{a}_0} \equiv f_0(x) + \sum_{j=1}^{m} \delta a_j \frac{\partial f_0(x)}{\partial a_j} \qquad (4.28)$$

In dieser Näherung ist χ^2 eine Funktion der Parametervariation δa_j

$$\chi^2(\delta\vec{a}) = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - f_0(x_i) - \sum_{j=1}^{m} \delta a_j \frac{\partial f_0(x_i)}{\partial a_j} \right]^2 \qquad (4.29)$$

Mit den Definitionen

$$y_i' \equiv y_i - f_0(x_i)\,, \quad a_j' \equiv \delta a_j \quad \text{und} \quad h_j'(x_i) \equiv \partial f_0(x_i)/\partial a_j \qquad (4.30)$$

ist dies formal identisch mit dem χ^2-Fit mit einer linearen Funktion (Gln. 4.1 und 4.2). Die Komponenten des m-dimensionalen Vektors $\vec{b}$ und der $m \times m$-Matrix S sind damit (Gl. 4.6)

$$\boxed{b_k = \sum_{i=1}^{n} g_i \left[y_i - f_0(x_i)\right] \frac{\partial f_0(x_i)}{\partial a_k}} \quad \text{und} \quad \boxed{S_{jk} = \sum_{i=1}^{n} g_i \frac{\partial f_0(x_i)}{\partial a_j} \frac{\partial f_0(x_i)}{\partial a_k}} \qquad (4.31)$$

Die Lösung erfolgt durch Matrizen-Inversion (Gl. 4.7)

$$\boxed{\delta\vec{a} = S^{-1}\,\vec{b} = C\,\vec{b}} \qquad (4.32)$$

Damit erhält man aus einer ersten Schätzung der Parameter $\vec{a}_0$ eine bessere $\vec{a}_0 + \delta\vec{a}$, die für den nächsten Iterationsschritt benutzt werden kann.

Diese Methode ist sehr gut, wenn man schon nahe beim χ^2-Minimum ist, d.h. die erste Schätzung der Parameter $\vec{a}_0$ ist schon nahe an der besten Schätzung. Ist man weit weg vom Minimum, so ist die Gradienten-Methode (siehe z.B. [Bevi 92]) besser geeignet. Die Marquardt-Methode verbindet beide Methoden mit einem Steuerparameter λ. Die Diagonalelemente der S-Matrix werden modifiziert

$$S_{jj}' = S_{jj}\,(1+\lambda)\,, \quad S_{jk}' = S_{jk} \ \text{ für } \ k \neq j\,. \qquad (4.33)$$

Bei sehr kleinem λ ändert sich praktisch nichts und wir haben die oben beschriebene Methode (Linearisierung der Fitfunktion). Ist λ sehr gross, so dominieren die Diagonalelemente und wir haben die Gradienten-Methode.

Der Algorithmus von Marquardt ist

1. Berechne $\chi^2(\vec{a})$ mit $\vec{a} = \vec{a_0}$.
2. Start mit $\lambda = 10^{-3}$.
3. Berechne $\delta\vec{a}$ und $\chi^2(\vec{a} + \delta\vec{a})$ mit diesem λ-Wert.
4. Wenn $\chi^2(\vec{a} + \delta\vec{a})$ grosser ist als $\chi^2(\vec{a})$, so multipliziere λ mit einem Faktor 10 und wiederhole Schritt 3.
5. Wenn $\chi^2(\vec{a} + \delta\vec{a})$ kleiner ist als $\chi^2(\vec{a})$, so dividiere λ durch einen Faktor 10. Ersetze $\vec{a}$ durch die neue Schätzung $\vec{a} + \delta\vec{a}$ und wiederhole Schritt 3.

Die Iteration wird so lange wiederholt bis der Wert von χ^2 sich praktisch nicht mehr ändert.

Dieser Algorithmus wird mit der MATLAB-Funktion `LSminimum` berechnet, wobei Schritt 3 mit der MATLAB-Funktion `mrqmin` berechnet wird (Programm 4.7; siehe auch die MATLAB-Funktion `polfit` in Programm 4.2). Der Benutzer hat eine MATLAB-Funktion zu schreiben, die die Fitfunktion und ihre Ableitungen nach den Parametern berechnet.

```
%------------------------------------------------------------------------
%    MATLAB program 4.7: non linear least squares fit
%    -----------------------------------------------

function a = LSminimum(data,fnc,par,par_desc,x,l)
%-----------------------------------------------
%   LSminimum(data,fnc,par,par_desc,x,l): non linear least squares fit
%                                         using the function mrqmin
%   input:  array data with 2 or 3 columns: x(i), y(i), (sig(i) optional)
%           fitfunction and derivatives: fnc; parameters: par
%           par_desc: string (optional);  x: x-values to plot fnc(x)
%           (optional);  l: l = 1 : semilog plot (optional)
%   output: results of each iteration are plotted and printed
%           final values of parameters and errors: a
if nargin == 3
   par_desc = '               p a r a m e t e r s                    ';
```



```
   x = data(:,1);                                l = 0;
elseif nargin == 4
   x = data(:,1);                                l = 0;
elseif nargin == 5,                              l = 0;
end
if l == 1,                                       logplot_data(data)
else,                                            plot_data(data)
end
grid,                                            k = 1;
itst = 0;                                        lambda = 1.0e-3;
fx = feval(fnc,x,par);
if l == 1,                                       semilogy(x,fx(:,1),'g-')
else,                                            plot(x,fx(:,1),'g-')
end
m = length(par);
while (itst < 2) & (k < 20)
   a = mrqmin(data,fnc,par,lambda);              % next iteration
   ochisq = a(2*m+1);                            chisq  = a(2*m+2);
   if k == 1
      fprintf('Start values:    chi2: %10.4f\n',ochisq);
      fprintf([par_desc '\n']);
      for i = 1:m,                               fprintf('%8.2e  ',par(i));
      end;                                       fprintf('\n\n');
   end
   fprintf('Iteration#  %2.0f',k);               k = k+1;
   fprintf('   chi^2: %10.4f  lambda: %9.2e\n',chisq,lambda);
   if chisq > ochisq                             % did trial succeed?
      itst = 0;
      fprintf('Bad step; try new one with increased lambda value');
      lambda = 10*lambda;                        % no, increase lambda
   else
      lambda = 0.1*lambda;                       % yes, new parameters
      par = a(1:m);                              fprintf([par_desc '\n']);
      for i = 1:m,                               fprintf('%8.2e  ',par(i));
      end;                                       fx = feval(fnc,x,par);
      if l == 1,                                 semilogy(x,fx(:,1),'k--')
      else,                                      plot(x,fx(:,1),'k--')
      end;
   end;                                          fprintf('\n\n');
   if abs(ochisq - chisq) < 0.001                % near minimum?
      itst = itst + 1;
   end
end;
```

```
if l == 1,                                      semilogy(x,fx(:,1),'b-')
else,                                           plot(x,fx(:,1),'b-')
end;
a = mrqmin(data,fnc,par,0);
%-------------------------------------------------------------------------

function z = mrqmin(data,func,a,lambda)
%-------------------------------------
%   function z = mrqmin(data,func,a,lambda): (weighted) least squares fit
%   using Marquardts method with parameter lambda; algorithm taken from
%   "Numerical Recipes", W.H.Press et al.;
%   calculates one step in the iteration
%   input:  array data with 2 or 3 columns: x(i), y(i), (sig(i) optional)
%           fitfunction (string): func; parameters (m-dim. vector): a
%   output: new estimates for parameters z(1:m) and errors z(m+1:2*m),
%           old and new value of chisquare z(2*m+1:2*m+2),
%           degrees of freedom and probability z(2*m+3:2*m+4),
%           last call with lambda = 0 prints full results
[n m] = size(a);
if n < m,                                       a = a';
end
x = data(:,1);                                  y = data(:,2);
n =length(x);                                   sd = size(data);
if sd(2) > 2,                                   sig = data(:,3);
else,                                           sig = ones(n,1);
end;                                            g = 1 ./(sig.*sig);
hx = feval(func,x,a);                           % function values hx(:,1)
                                                % and derivatives hx(:,2:m+1)
S = zeros(m,m);                                 b = zeros(m,1);
for k = 1:m
    for l = 1:m                                 % calculate matrix
        S(k,l) = sum(g.*hx(:,k+1).*hx(:,l+1));
    end;
    S(k,k) = (1 + lambda)*S(k,k);               % increase diagonal elements
    b(k,1) = sum(g.*hx(:,k+1).*(y - hx(:,1)));      % calculate vector
end;
C = inv(S);                                     % covariance matrix
da = C*b;
a = a + da;                                     % new parameters
old_chi2 = sum(g.*(y - hx(:,1)).^2);            % old value of chisquare
hx = feval(func,x,a);                           % function values hx(:,1)
chi2 = sum(g.*(y - hx(:,1)).^2);                % new value of chisquare
prob = Pchisqr(n-m,chi2);                       % probability
```



```
z = [a' sqrt(diag(C))' old_chi2 chi2 n-m prob];
if lambda == 0                                  % print out full results
   disp('parameter +- error'),                  format short e
   disp([a sqrt(diag(C))]),                     format short
   for k = 1:m                                  % calculate correlation matrix
       for l = 1:m
           rho(k,l) = C(k,l)/sqrt(C(k,k)*C(l,l));
       end;
   end;
   disp('correlation matrix'),                  disp(rho)
   disp('chisquare, dof, probability'),         disp([chi2 n-m prob])
end
%-------------------------------------------------------------------------
```

4.2.3 Beispiel 1: Gauss-Peak mit quadratischem Untergrund

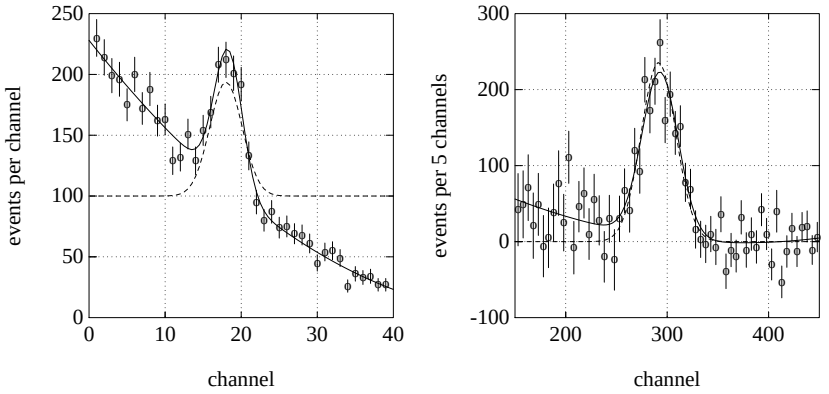

Abbildung 4.7: Anpassung eines Gauss-Peaks mit quadratischem Untergrund (- - Startwerte; — beste Schätzwerte). Links: Monte-Carlo-Daten; Rechts: Daten aus dem VP-Versuch "Compton-Streuung".

Die Auswertung eines Peaks (die Bestimmung der Position und der Fläche) ist eine häufige Aufgabe der Datenanalyse. Ein Beispiel mit einem gemessenen Untergrund haben wir im Kap. 1.3.2 (Abb. 1.4, Programm 1.5) besprochen. In Abb. 4.7 ist dieses Beispiel nochmals mit der Fitfunktion 4.34 dargestellt.

Wie die nicht-lineare Anpassung eines Gauss-Peaks mit quadratischem Untergrund durchgeführt wird, wollen wir an den Monte-Carlo-Daten des

Programms 3.4 demonstrieren. Die lineare Anpassung des Untergrundes und die Bestimmung der Fläche des Peaks wurden schon im Kap. 4.1.3 (Programm 4.2, Abb. 4.2) besprochen. Dabei war das Resultat etwas abhängig von den gewählten Grenzen des Peaks, aber unabhängig von einer angenommenen Form des Peaks. Ist die Form des Peaks aber bekannt, so können Untergrund und Peak auf einmal bestimmt werden. Im unserem Beispiel ist dies der Fall und die Fitfunktion ist

$$\begin{aligned} f(x;\vec{a}) &= a_1 + a_2 x + a_3 x^2 + \frac{a_4}{\sqrt{2\pi}\,a_6}\exp\left[-0.5(x-a_5)^2/a_6^2\right] \\ &\equiv a_1 + a_2 x + a_3 x^2 + g(x) \end{aligned} \tag{4.34}$$

Für den nicht-linearen Fit mit dem Algorithmus von Marquardt (Kap. 4.2.2) müssen auch die Ableitungen $\partial f(x;\vec{a})/\partial a_k$ bekannt sein. Die Ableitungen der linearen Parameter $a_1 \cdots a_4$ ist trivial (siehe auch Kap. 4.1.4, Programm 4.3). Für die nicht-linearen Parameter a_5 und a_6 sind sie

$$\frac{\partial f(x;\vec{a})}{\partial a_5} = \frac{x-a_5}{a_6^2}\,g(x) \qquad \frac{\partial f(x;\vec{a})}{\partial a_6} = \left(\frac{(x-a_5)^2}{a_6^3} - \frac{1}{a_6}\right) g(x) \tag{4.35}$$

Die Fitfunktion und ihre Ableitungen nach den Parametern ist in der MATLAB-Funktion `fx = gb_fdfda(x,a)` programmiert (Programm 4.8). Mit einer groben Schätzung der Parameter als Startwert `FWHM = 5; par = [100 0 0 100*FWHM 18 FWHM/2.35]` kann jetzt die χ^2-Minimalisierung durchgeführt werden `LSminimum(gb_data,'gb_fdfda',par)` (die letzten 2 der 5 Funktionsargumenten können als Option für eine schöne Darstellung spezifiziert werden). Das Minimum wird nach 5 Iterationsschritten erreicht (Programm 4.8; Abb. 4.7). Die Schätzwerte der linearen Parameter $a_1 = 228 \pm 7$, $a_2 = -7.9 \pm 0.7$, $a_3 = 0.069 \pm 0.015$, $a_4 = N = 527 \pm 50$ sind in Übereinstimmung mit dem Resultat des Programms 4.2. Zusätzlich erhalten wir jetzt auch die Position $a_5 = 18.3 \pm 0.2$ und die Breite FWHM $= 2.35 * a_6 = 4.3 \pm 0.4$ des Gauss-Peaks.

```
%-------------------------------------------------------------------------
%    program 4.8: fit of gaussian + quadratic background
%    ---------------------------------------------------
load gb_data;
gb_data = [gb_data sqrt(gb_data(:,2))];
FWHM = 5;
par = [100 0 0 100*FWHM 18 FWHM/2.35];
par_desc = '  a[1]      a[2]      a[3]      a[4]      a[5]      a[6]';
```



```
LSminimum(gb_data,'gb_fdfda',par,par_desc,(0:0.5:40));
%-------------------------------------------------------------------------

function fx = gb_fdfda(x,a)
%--------------------------
%    gaussian + quadratic background
%    returns function value and derivatives df/da, parameters a
%-------------------------------------------------------------------
[n m] = size(x);
if n > m,                x = x';              end
bac = a(1) + a(2)*x + a(3)*x.^2;              arg = (x - a(5))/a(6);
gx  = a(4)*exp(-0.5*arg.^2)/(sqrt(2*pi)*a(6));
dfda6 = gx.*(arg.^2 - 1)/a(6);                dfda5 = gx.*arg/a(6);
fx = [bac+gx; ones(size(x)); x;  x.*x; gx/a(4); dfda5; dfda6]';
%-------------------------------------------------------------------------

%    output of function LSminimum:
%    -----------------------------
%    Start values:    chi2:  1740.8104
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    1.00e+02  0.00e+00  0.00e+00  5.00e+02  1.80e+01  2.13e+00
%
%    Iteration#   1   chi^2:    37.0177  lambda:  1.00e-03
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    2.14e+02  -6.19e+00  3.17e-02  4.68e+02  1.83e+01  1.65e+00
%
%    Iteration#   2   chi^2:    30.0115  lambda:  1.00e-04
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    2.27e+02  -7.86e+00  6.85e-02  5.29e+02  1.83e+01  1.89e+00
%
%    Iteration#   3   chi^2:    29.8970  lambda:  1.00e-05
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    2.28e+02  -7.87e+00  6.86e-02  5.25e+02  1.83e+01  1.83e+00
%
%    Iteration#   4   chi^2:    29.8907  lambda:  1.00e-06
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    2.28e+02  -7.88e+00  6.90e-02  5.27e+02  1.83e+01  1.85e+00
%
%    Iteration#   5   chi^2:    29.8903  lambda:  1.00e-07
%      a[1]      a[2]      a[3]      a[4]      a[5]      a[6]
%    2.28e+02  -7.88e+00  6.89e-02  5.26e+02  1.83e+01  1.85e+00
%
%    output of function mrqmin:
```

```
%    -------------------------
%    parameter +- error
%       2.2765e+02   6.5161e+00
%      -7.8813e+00   6.7628e-01
%       6.8965e-02   1.5122e-02
%       5.2652e+02   5.0118e+01
%       1.8333e+01   1.6489e-01
%       1.8462e+00   1.6926e-01
%    correlation matrix
%    1.00  -0.86   0.74   0.06   0.10   0.04
%   -0.86   1.00  -0.98  -0.35  -0.09  -0.25
%    0.74  -0.98   1.00   0.43   0.07   0.31
%    0.06  -0.35   0.43   1.00  -0.05   0.58
%    0.10  -0.09   0.07  -0.05   1.00  -0.07
%    0.04  -0.25   0.31   0.58  -0.07   1.00
%    chisquare, dof, probability
%       29.8903   33.0000    0.6227
%-------------------------------------------------------------------------
```

4.2.4 *Beispiel 2: Exponentialfunktion gefaltet mit einer Gauss-Funktion

Auch die nicht-lineare Anpassung einer Exponentialverteilung (meistens eine Zeitverteilung), die mit einer Gauss-Verteilung (Auflösungsfunktion) gefaltet ist, ist eine häufige Aufgabe der Datenanalyse, z.B. im VP-Versuch "Positronium". Zur Demonstration des Marquardt-Verfahrens (Kap. 4.2.2) werden wir Monte-Carlo-Daten (t_i, $y_i \pm \sqrt{y_i}$) benützen (Programm 3.10). Die Fitfunktion ist (Gl. 3.48)

$$f(t;\, t_0,\, \lambda,\, \sigma) = \frac{\lambda}{2}\, \exp\left(-\lambda\left[t - t_0 - \lambda\sigma^2/2\right]\right) \times \left[1 + \mathrm{erf}\left(\frac{t - t_0 - \lambda\sigma^2}{\sqrt{2}\,\sigma}\right)\right] \qquad (4.36)$$

Die drei Parameter sind: die Zerfallskonstante λ, die Zeitauflösung σ und der Zeitnullpunkt t_0. Mit der Kettenregel und mit

$$\frac{d\mathrm{erf}(u)}{du} = \frac{2}{\sqrt{\pi}}\, e^{-u^2} \qquad u = \frac{t - t_0 - \lambda\sigma^2}{\sqrt{2}\,\sigma}$$

erhalten wir die Ableitungen nach den Parametern

$$\frac{\partial f(t)}{\partial \lambda} \;=\; f(x)(1/\lambda - t + t_0 + \lambda\sigma^2) - g(t)\,\sigma$$

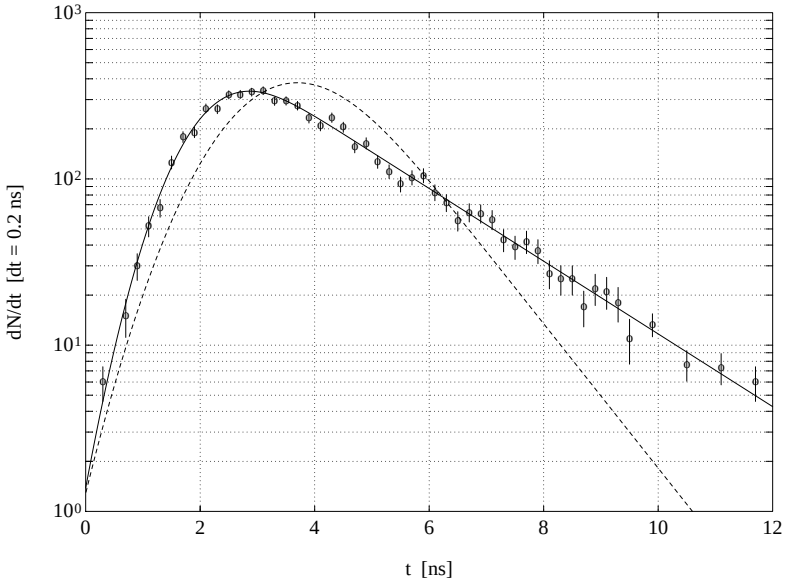

Abbildung 4.8: Anpassung einer Exponentialverteilung, die mit einer Gauss-Verteilung gefaltet ist, an Monte-Carlo-Daten (- - Startwerte; — beste Schätzwerte).
protect

$$\begin{aligned} \frac{\partial f(t)}{\partial \sigma} &= \lambda^2\sigma f(t) - [\lambda + (t - t_0)/\sigma^2]g(t) \qquad (4.37)\\ \frac{\partial f(t)}{\partial t_0} &= \lambda f(t) - g(t)/\sigma, \quad g(t) = \frac{\lambda}{\sqrt{2\pi}}\, \exp\left(-\lambda\left[t - t_0 - \lambda\sigma^2/2\right] - u^2\right) \end{aligned}$$

Die Fitfunktion und ihre Ableitungen nach den Parametern ist in der MATLAB-Funktion `fx = EGfunction(x,p)` programmiert (Programm 4.9). Die Fitfunktion wird mit der Gesamtzahl der Ereignisse normiert: `norm = sum(y)`. In einer zweiten Iteration wird für die endliche Messzeit korrigiert ($1 - \exp(-\lambda T) \simeq 0.98$). Man hätte die Normierung auch als zusätzlichen Parameter in der Fitfunktion definieren können (siehe auch die Diskussion in Kap. 4.2.1). Die Werte $y_i \leq 10$ (`y=[EGdata(1:3,2); EGdata(49:60,2)]`) werden in Dreiergruppen zusammengefasst: `y3 = sum(reshape(y,3,5))` und entsprechend werden die Fehler

und die t-Werte (x-Werte) berechnet. Mit einer groben Schätzung der Parameter als Startwert `[1 1 3]` kann jetzt die χ^2-Minimalisierung durchgeführt werden `LSminimum(data,'EGfunction',par,par_names,x,1)`. Das Minimum wird nach 6 Iterationsschritten erreicht (Programm 4.9; Abb. 4.8), wobei die ersten beiden Schritten keinen Erfolg hatten und der Steuerparameter λ (*nicht* die Zerfallskonstante λ) um einen Faktor 10 erhöht werde musste (siehe Punkt 4 im Marquardt-Algorithmus, Kap. 4.2.2). Die Schätzwerte der Parameter, $\lambda = (0.504 \pm 0.010)$ 1/ns, $\sigma = (0.719 \pm 0.021)$ ns und $t_0 = (2.017 \pm 0.026)$ ns, sind in Übereinstimmung mit den Monte-Carlo-Eingabewerten (0.5 /ns, 0.7 ns bzw. 2 ns).

Bei dieser relativ komplizierten Fitfunktion ist es notwendig, die Programmierung der Funktion und ihre Ableitungen mit einem Hilfsprogramm (Programm `fx = EGderivatives` in Programm 4.9) zu überprüfen (dies ist ein allgemeines Rezept für erfolgreiches Programmieren: Ein komplexes Programm wird in übersichtliche, geprüfte Module zerlegt).

```
%-------------------------------------------------------------------------
%    program 4.9: fit of exponential folded with gaussian
%    ----------------------------------------------------
load EGdata;
x    = [EGdata(1:3,1); EGdata(49:60,1)];
y    = [EGdata(1:3,2); EGdata(49:60,2)];
x3   = sum(reshape(x,3,5));
y3   = sum(reshape(y,3,5));
x    = [EGdata(4:48,1)' x3/3];
y    = [EGdata(4:48,2)' y3/3];
sig  = [sqrt(EGdata(4:48,2))' sqrt(y3)/3];
data = [x' y' sig'];
x    = 0:0.05:12;
par_names = ' lambda     sigma     x0';
% par = [0.5 2.5/3 2.5];
par  = [1 1 3];
LSminimum(data,'EGfunction',par,par_names,x,1);
axis([0 12 1 1000])
%-------------------------------------------------------------------------

function fx = EGfunction(x,p)
%---------------------------
[n m] = size(x);
if n < m,               x = x';                   end
```



```
lambda = p(1);          sigma = p(2);             x0 = p(3);
norm = 1211;            % norm = 1191 = sum(y)
u    = (x - x0)/(sqrt(2)*sigma) - lambda*sigma/sqrt(2);
fx   = 0.5*lambda*exp(-lambda*(x - x0 - 0.5*lambda*sigma^2));
gx   = sqrt(2/pi)*fx .* exp(-u.^2);
fx   = (1 + erf(u)).*fx;
dfdl = (1/lambda + lambda*sigma^2 - x + x0).*fx - sigma*gx;
dfds = lambda^2*sigma*fx - (lambda + (x - x0)/sigma^2).*gx;
dfdx0= lambda*fx - gx/sigma;
factor = 1 - exp(-lambda*(12 - x0));  norm = norm/factor;
dfdl = dfdl - (12 - x0)*exp(-lambda*(12 - x0))*fx/factor;
dfdx0 = dfdx0 + lambda*exp(-lambda*(12 - x0))*fx/factor;
fx   = [norm*fx norm*dfdl norm*dfds norm*dfdx0];
%-------------------------------------------------------------------------

%    output of function LSminimum:
%    -----------------------------
%    Start values:    chi2:  1164.2252
%     lambda     sigma     x0
%    1.00e+00  1.00e+00  3.00e+00
%
%    Iteration#   1   chi^2:  4616.0266  lambda:  1.00e-03
%    Bad step; try new one with increased lambda value
%
%    Iteration#   2   chi^2:  2690.5300  lambda:  1.00e-02
%    Bad step; try new one with increased lambda value
%
%    Iteration#   3   chi^2:   356.7286  lambda:  1.00e-01
%     lambda     sigma     x0
%    5.88e-01  1.11e+00  2.42e+00
%
%    Iteration#   4   chi^2:    70.1708  lambda:  1.00e-02
%     lambda     sigma     x0
%    4.64e-01  7.29e-01  1.97e+00
%
%    Iteration#   5   chi^2:    45.3784  lambda:  1.00e-03
%     lambda     sigma     x0
%    5.01e-01  7.19e-01  2.02e+00
%
%    Iteration#   6   chi^2:    45.2582  lambda:  1.00e-04
%     lambda     sigma     x0
%    5.04e-01  7.19e-01  2.02e+00
%
```

```
%    parameter +- error
%       5.0429e-01   1.0256e-02
%       7.1938e-01   2.1118e-02
%       2.0179e+00   2.6471e-02
%
%    correlation matrix
%       1.00   0.48   0.68
%       0.48   1.00   0.61
%       0.68   0.61   1.00
%    chisquare, dof, probability
%       45.2582   47.0000    0.5449
%-------------------------------------------------------------------------

%     EGderivatives: compares analytical and numerical derivatives
%     ------------------------------------------------------------
lambda = 0.5;                  sigma = 0.7;                    x0 = 2.0;
x = 0:2:12;                    out = EGfunction(x,[lambda sigma x0]);
out1 = EGfunction(x,[lambda-0.001,sigma x0]);
out2 = EGfunction(x,[lambda+0.001,sigma x0]);
dfdla = out(:,2);              dfdln = (out2(:,1) - out1(:,1))/0.002;
out1 = EGfunction(x,[lambda,sigma-0.001 x0]);
out2 = EGfunction(x,[lambda,sigma+0.001 x0]);
dfdsa = out(:,3);              dfdsn = (out2(:,1) - out1(:,1))/0.002;
out1 = EGfunction(x,[lambda,sigma x0-0.001]);
out2 = EGfunction(x,[lambda,sigma x0+0.001]);
dfdx0a = out(:,4);             dfdx0n = (out2(:,1) - out1(:,1))/0.002;
fprintf('\n x   f(x)     df/dlambda       df/dsigma        df/dx0\n')
fprintf('              ana     num     ana     num     ana     num')
for i = 1:length(x)
   fprintf('\n%2.0f %7.2f %7.2f ',x(i),out(i,1),dfdla(i))
   fprintf('%7.2f %7.2f %7.2f ',dfdln(i),dfdsa(i),dfdsn(i))
   fprintf('%7.2f %7.2f',dfdx0a(i),dfdx0n(i))
end
%-------------------------------------------------------------------------

%     x   f(x)     df/dlambda       df/dsigma        df/dx0
%                  ana     num     ana     num     ana     num
%      0    1.18    2.15    2.14   14.81   14.86   -5.24   -5.25
%      2  233.79  355.77  355.80  -79.87  -79.84 -228.19 -228.13
%      4  235.38   54.81   54.81   22.51   22.50  111.86  111.86
%      6   87.12 -152.90 -152.90   15.25   15.25   43.56   43.56
%      8   32.05 -120.35 -120.35    5.61    5.61   16.03   16.03
%     10   11.79  -67.86  -67.86    2.06    2.06    5.90    5.90
```



```
%     12    4.34  -33.64  -33.64    0.76    0.76    2.17    2.17
```

4.2.5 *Simplex-Methode

Neben der Linearisierung der Fitfunktion (Marquardt-Methode, Kap. 4.2.2) gibt es weitere Methoden der nicht-linearen Anpassung (siehe z.B. [Bevi 92] und [Pres 88]). Eine davon, die sogenannte Simplex-Methode ([Bran 99], [Pres 88]), wird auch von den MATLAB-Funktion `fmins` benützt. Der Vorteil dieser Methode ist, dass sie keine Ableitungen der Fitfunktion benötigt. Damit gibt sie aber auch keine Fehler der Parameter. Ein Beispiel, das wir schon mit der Marquardt-Methode berechnet haben (Programm 4.9), ist im Programm 4.10 gegeben. Der Benützer hat die Funktion, die minimalisiert werden sollte, zu programmieren und eine erste Schätzung der Parameter zu geben. In unserem Beispiel ist die Funktion eine χ^2-Funktion, `chi2 = EGchi2(par)` und mit der Schätzung `a = [1 1 3]'` wird die Simplex-Minimierung `a = fmins('EGchi2',a)` durchgeführt. Das Minimum wird erst nach 60 Iterationsschritten erreicht; das Resultat (Programm 4.10) ist das gleiche wie vorher (Programm 4.9). Man beachte, dass die Daten als globale Variablen deklariert sind (`global Gdata`); sie stehen damit auch der χ^2-Funktion zu Verfügung

Die **Fehler der Parameter** können grafisch ermittelt werden, was nur für wenige oder nicht korrelierte Parameter möglich ist, oder mit den Ableitungen von χ^2 nach den Parametern (siehe Kap. 4.1.1, Gl. 4.9)

$$\boxed{\frac{\partial^2\chi^2(\vec{a})}{\partial a_j \partial a_k} = 2S_{jk} = 2\sum_{i=1}^{n} g_i \frac{\partial f(x_i;\vec{a})}{\partial a_j}\frac{\partial f(x_i;\vec{a})}{\partial a_k}} \tag{4.38}$$

Dazu brauchen wir aber wieder die Ableitungen der Fitfunktion nach den Parametern (wie vorher, bei der Linearisierung der Fitfunktion, haben wir die quadratischen Ableitungen vernachlässigt). Die Kovarianzmatrix ist $C = S^{-1}$. Diese Methode wird im Programm 4.10 zur Berechnung der Fehler benützt.

```
%-------------------------------------------------------------------------
%    MATLAB program 4.10: non linear least squares fit (simplex method)
%    -------------------------------------------------------------------
load EGdata;
x = [EGdata(1:3,1); EGdata(49:60,1)];
y=[EGdata(1:3,2); EGdata(49:60,2)];
x3 = sum(reshape(x,3,5));
y3 = sum(reshape(y,3,5));
x = [EGdata(4:48,1)' x3/3];
```

```
y = [EGdata(4:48,2)' y3/3];
sig = [sqrt(EGdata(4:48,2))' sqrt(y3)/3];
Gdata = [x' y' sig'];
a = [1 1 3]';
global Gdata
a = fmins('EGchi2',a,[1 0.01 0.1]);
chi2 = EGchi2(a);
fprintf('\nlambda = %5.3f,  sigma = %5.3f,  ',a(1),a(2))
fprintf('x0 = %5.3f,  chi^2 = %4.1f\n',a(3),chi2)

%       calculation of  errors
%       ----------------------
fx = EGfunction(x,a);                        g = 1 ./ sig'.^2;
S = zeros(length(a),length(a));
for j = 1:length(a)
   for k = j:length(a)
      S(j,k) = sum(g.*fx(:,k+1).*fx(:,j+1)); S(k,j) = S(j,k);
   end
end;                                         C = inv(S);
fprintf('\nsig_lambda = %5.3f,  sig_sigma = ',sqrt(C(1,1)))
fprintf('%5.3f,  sig_x0 = %5.3f\n',sqrt(C(2,2)),sqrt(C(3,3)))

%-------------------------------------------------------------------------
%   lambda     = 0.502,  sigma     = 0.727,  x0     = 2.025,  chi^2 = 46
%   sig_lambda = 0.010,  sig_sigma = 0.021,  sig_x0 = 0.027
%-------------------------------------------------------------------------

function chi2 = EGchi2(par)
%--------------------------
x = Gdata(:,1);                              y = Gdata(:,2);
sig = Gdata(:,3);
fx = EGfunction(x,par);                      fx1 = fx(:,1);
chi2 =sum((y - fx1).^2 ./sig.^2);
%-------------------------------------------------------------------------
```