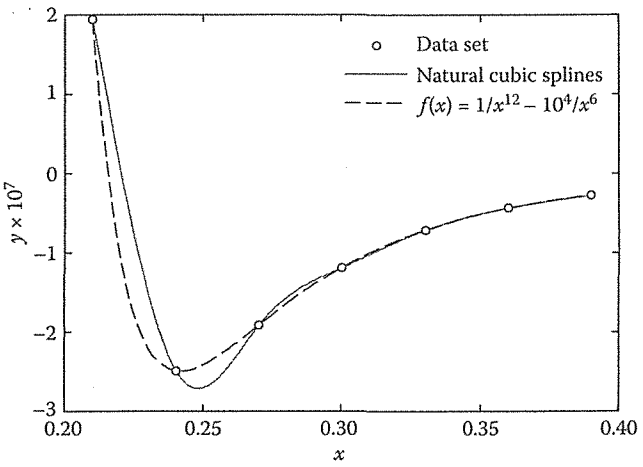

FIGURE 9.4 Case in which the natural cubic spline deviates significantly from the modeled function due to insufficient interpolated points. The interpolated data samples the function $f(x) = 1/x^{12} - 10^4/x^6$.

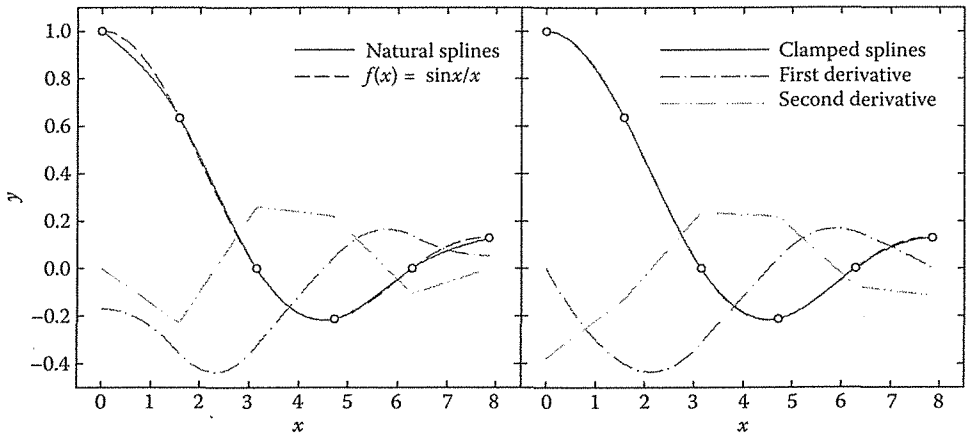

FIGURE 9.5 Interpolation with natural cubic splines (left) and clamped cubic splines (right). The modeled dependence, $f(x) = \sin x/x$, is sampled in the interval $[0, 5\pi/2]$ with a $\pi/2$ spacing.

the true behavior of the modeled function. Quite in contrast, *clamped boundaries* clearly improve the matching between the interpolant and the modeled dependence (see the right panel of Figure 9.5).

9.5 Linear Regression

In quite a few practical situations, the plot of the observations or the theoretical framework on which the data collection is based suggests a linear functional dependence of the modeled function on the independent variable (see Figure 9.6):

$$F(x; a, b) = ax + b. \tag{9.40}$$

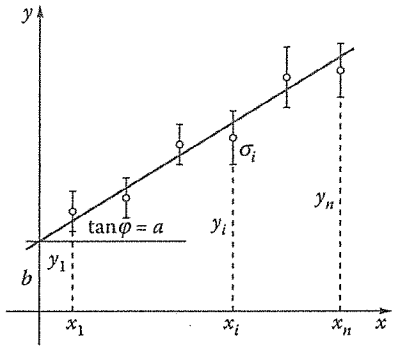

FIGURE 9.6 Linear regression based on the least-squares approximation. The parameter a of the model function represents the slope of the regression line, while b is the corresponding y-intercept.

Linear regression actually represents the simplest application of the general concepts regarding the least-squares fitting of model functions to tabulated dependences, which have been presented in Section 9.1.

Assuming the uncertainties σ_i associated with the observed values y_i to be available, the quality of the model function $F(x; a, b)$ adjusted to the observed data may be quantified by the χ^2-merit function (Equation 9.8), taking in this case the form:

$$\chi^2(a,b) = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}(y_i - ax_i - b)^2. \tag{9.41}$$

The optimal model parameters a and b, which minimize χ^2, result from the condition of vanishing partial derivatives of χ^2 with respect to a and b:

$$\begin{aligned} \frac{\partial \chi^2}{\partial a} &\equiv -2\sum_{i=1}^{n} \frac{1}{\sigma_i^2}(y_i - ax_i - b)x_i = 0, \\ \frac{\partial \chi^2}{\partial b} &\equiv -2\sum_{i=1}^{n} \frac{1}{\sigma_i^2}(y_i - ax_i - b) = 0. \end{aligned} \tag{9.42}$$

By introducing the notations for the involved sums

$$s = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}, \quad s_x = \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}, \quad s_{xx} = \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2}, \quad s_y = \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2}, \quad s_{xy} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2}, \tag{9.43}$$

one obtains the simplified system:

$$\begin{cases} s_{xx}a + s_x b = s_{xy} \\ s_x a + s\, b = s_y. \end{cases} \tag{9.44}$$

With the additional notation

$$\Delta = s\, s_{xx} - s_x^2, \tag{9.45}$$

the solution of the system provides the values of the optimized model parameters:

$$a = \frac{s\, s_{xy} - s_x s_y}{\Delta}, \quad b = \frac{s_y s_{xx} - s_x s_{xy}}{\Delta}. \tag{9.46}$$

To evaluate the probable imprecisions associated with the optimal model parameters (9.46), one can apply the general definition (9.11) of the variance:

$$\sigma_a^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\frac{\partial a}{\partial y_i}\right)^2, \quad \sigma_b^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\frac{\partial b}{\partial y_i}\right)^2.$$

By using the specific expressions (9.46) of the parameters, one obtains the derivatives:

$$\frac{\partial a}{\partial y_i} = \frac{s\,x_i - s_x}{\sigma_i^2 \Delta}, \quad \frac{\partial b}{\partial y_i} = \frac{s_{xx} - s_x x_i}{\sigma_i^2 \Delta}.$$

Summation over all data points yields, for example, for the variance of the parameter a:

$$\sigma_a^2 = \sum_{i=1}^{n} \frac{s^2 x_i^2 - 2s\,x_i s_x + s_x^2}{\sigma_i^2 \Delta^2} = \frac{s^2 s_{xx} - 2s\,s_x^2 + s\,s_x^2}{\Delta^2} = \frac{s(s\,s_{xx} - s_x^2)}{\Delta^2} = \frac{s}{\Delta}.$$

Proceeding similarly for the parameter b, one finally obtains the usable expressions of the probable imprecisions of the two model parameters:

$$\sigma_a = \sqrt{\frac{s}{\Delta}}, \quad \sigma_b = \sqrt{\frac{s_{xx}}{\Delta}}. \tag{9.47}$$

The developed formalism of linear regression can be also applied to cases of nonlinear functional dependences which admit analytic inverses. For example, the exponential model

$$y = be^{ax}$$

can be obviously linearized by applying the logarithm and naming $y' = \ln y$ and $b' = \ln b$:

$$y' = ax + b'.$$

The adjustment of the parameters a and b' of the linearized model is to be carried out with respect to the transformed observations $y_i' = \ln y_i$ and the optimized parameter b of the initial nonlinear model finally results as $b = \exp(b')$.

Based on the `n` observed coordinate pairs, received by way of the arrays `x` and `y`, and the associated standard deviations, `sigmy`, the routine `LinFit` shown in Listing 9.8 returns the adjusted parameters `a` and `b` of the linear model, as well as the associated probable imprecisions, `sigma` and `sigmb`. As a synthetic measure of the fit quality, the routine also returns by `chi2` the value of the χ^2-merit function calculated with the optimized model parameters.

If the control parameter `iopt` is set to 0, the routine initializes all the components of `sigmy` with 1 and the procedure falls back to the least-squares approximation based on the simple sum of squared deviations as a merit function:

$$S = \sum_{i=1}^{n} (y_i - ax_i - b)^2. \tag{9.48}$$

Listing 9.8 Linear Regression (Python Coding)

```
#=============================================================================
def LinFit(x, y, sigmy, n, iopt):
#-----------------------------------------------------------------------------
#  Determines the coefficients a and b of a linear model P(x;a,b) = a * x + b
#  and the associated uncertainties, sigma and sigmb, for a set of observed
#  data points by "Chi-square" regression
#  x[]      - x-coordinates of observed data, i = 1,..,n
#  y[]      - y-coordinates of observed data
#  sigmy[]  - standard deviations of observed data
#  n        - number of observed data points
#  iopt     - iopt == 0 - initializes sigmy[i] = 1 (equal weights)
#           - iopt != 0 - uses the received values of sigmy[i]
#  a, b     - parameters of the linear model (output)
#  sigma    - uncertainties associated with the model parameters (output)
#  sigmb
#  chi2     - value of Chi-square merit function for the output parameters
#-----------------------------------------------------------------------------
   if (iopt == 0):
      for i in range(1,n+1): sigmy[i] = 1e0        # iopt = 0: equall weights

   s = sx = sy = sxx = sxy = 0e0                              # prepare sums
   for i in range(1,n+1):
      f = 1e0/(sigmy[i]*sigmy[i])
      s  += f
      sx += x[i] * f; sxx += x[i] * x[i] * f
      sy += y[i] * f; sxy += x[i] * y[i] * f

   f = 1e0/(s*sxx - sx*sx)
   a = (s *sxy - sx*sy ) * f; sigma = sqrt(s*f)          # model parameters
   b = (sy*sxx - sx*sxy) * f; sigmb = sqrt(sxx*f)        # and uncertainties

   chi2 = 0e0                                    # value of Chi-square function
   for i in range(1,n+1):
      f = (y[i] - a*x[i] - b)/sigmy[i]
      chi2 += f*f

   return (a, b, sigma, sigmb, chi2)
```

This option is typically useful in situations in which the standard deviations σ_i of the observed data are unknown. Anyway, it should be noted that the comparative use of the two merit functions, χ^2 and S, may lead to rather different regression lines.

In the example shown in Figure 9.7, the merit function S, not making use of observational errors σ_i and assigning equal weights to all observed data, yields a regression line not passing through all the error bars. In contrast, by applying the χ^2-regression and considering, for the sake of simplicity, errors σ_i proportional to the observed values y_i (a somewhat typical situation), the first data point, bearing the lowest imprecision, forces the regression line to pass also through its error bar.

Specifically, the coordinates of the tabulated data points are $(1, 0.8)$, $(2, 2.1)$, $(3, 2.8)$, $(4, 4.0)$, and $(5, 4.4)$, and the corresponding standard deviations are generated simply as $\sigma_i = 0.15\, y_i$. By using the functional S one obtains the model parameters $a = 0.9100$ and $b = 0.0900$, while for the χ^2-merit function, the values $a = 0.9983$ and $b = -0.1681$ result.

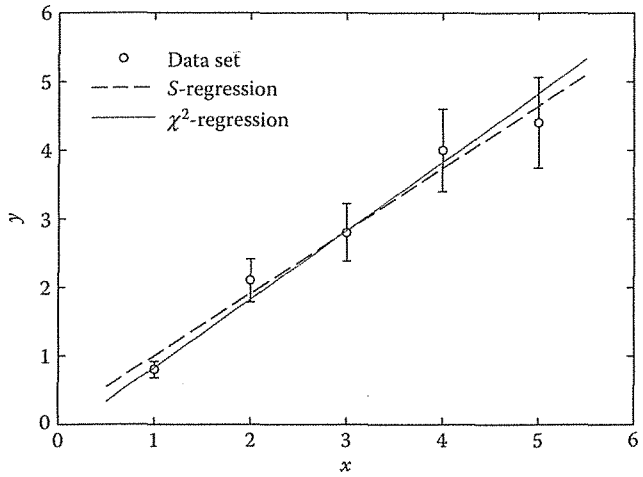

FIGURE 9.7 Example of linear regression evidencing the qualitative improvement of the fit based on the χ^2-merit function as compared to the simple sum of squared deviations, S.

9.6 Multilinear Regression Models

As a generalization of the linear regression, presented in the previous section, we consider in the following the *multilinear regression*, having as an important particular case the *polynomial regression.*

The task in the case of the multilinear regression is to fit a model represented by a *linear combination* of arbitrary functions of x,

$$F(x;\mathbf{a}) = \sum_{j=1}^{m} a_j F_j(x), \tag{9.49}$$

to a set of n observed data points, (x_i, y_i). The linear nature of the approach refers solely to the way in which the m parameters a_j enter the model. As for the *basis functions* $F_j(x)$, they have a fixed form not depending on the model parameters and can be arbitrarily nonlinear with respect to x.

The most widely used basis sets are the family of integer powers, $1, x, x^2, \ldots, x^{m-1}$, on which the polynomial regression is based, and the trigonometric functions $\sin kx$ and $\cos kx$ composing the Fourier series. Regarded as a particular case of polynomial regression, the linear regression involves only the basis functions 1 and x.

The χ^2-merit function for the multilinear regression has the form

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{m} a_j F_j(x_i) \right]^2, \tag{9.50}$$

where σ_i represent, as previously, the observational errors (standard deviations) associated with the observations y_i. The following formalism remains applicable also in the case of unknown errors by setting all σ_i equal, for example, $\sigma_i = 1$.

The optimal model parameters minimize the χ^2-merit function or, equivalently, zero its partial derivatives:

$$\frac{\partial \chi^2}{\partial a_k} \equiv -2 \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{m} a_j F_j(x_i) \right] F_k(x_i) = 0, \quad k = 1, 2, \ldots, m. \tag{9.51}$$

By changing the summation order and rearranging the terms, one obtains the so-called *system of normal equations* of the multilinear fitting problem, namely,

$$\sum_{j=1}^{m} c_{kj} a_j = b_k, \quad k = 1, 2, \ldots, m, \tag{9.52}$$

whose solutions are the searched-for optimal model parameters. The system matrix $\mathbf{C} = [c_{kj}]_{mm}$ and the vector of constant terms $\mathbf{b} = [b_k]_m$ are defined by

$$c_{kj} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} F_k(x_i) F_j(x_i), \tag{9.53}$$

$$b_k = \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2} F_k(x_i). \tag{9.54}$$

The estimates of the squared uncertainties (variances) associated with the optimized parameters can be obtained by applying the general formula

$$\sigma_{a_j}^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i} \right)^2. \tag{9.55}$$

To calculate the involved derivatives, we formally express the parameters a_j from the system of normal equations (9.52),

$$a_j = \sum_{k=1}^{m} c'_{jk} b_k = \sum_{k=1}^{m} c'_{jk} \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2} F_k(x_i),$$

where c'_{jk} are the elements of the inverse system matrix, $\mathbf{C}^{-1} = [c'_{jk}]_{mm}$. The derivatives of the parameters are hence

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{m} c'_{jk} \frac{1}{\sigma_i^2} F_k(x_i)$$

and, upon replacing them into the Formula 9.55 of the variance, inverting the summation order, and considering the definition (9.53) of the matrix elements c_{kj}, one successively obtains

$$\sigma_{a_j}^2 = \sum_{k,l=1}^{m} c'_{jk} c'_{jl} \sum_{i=1}^{n} \frac{1}{\sigma_i^2} F_k(x_i) F_l(x_i) = \sum_{k,l=1}^{m} c'_{jk} c'_{jl} c_{kl} = \sum_{l=1}^{m} c'_{jl} \delta_{jl}.$$

Here, we have also used the identity $\sum_{k=1}^{m} c'_{jk} c_{kl} = \delta_{jl}$, where δ_{jl} is Kronecker's delta. Consequently, the squared uncertainty of the optimized parameter a_j is given by

$$\sigma_{a_j}^2 = c'_{jj}, \tag{9.56}$$

that is, by the corresponding diagonal element c'_{jj} of the inverse matrix $\mathbf{C}^{-1}$ of the system of normal equations. In a broader sense, c'_{jk} represents the *covariance* associated with the pair of parameters (a_j, a_k) (Press et al. 2007).

The implementation of the described method of normal equations for the problem of multilinear regression is given in Listing 9.9. The function `MultiFit` receives the `n` coordinates of the observed

Listing 9.9 Multilinear Regression (Python Coding)

```
#=============================================================================
def MultiFit(x, y, sigmy, n, iopt, a, sigma, npar, Func):
#-----------------------------------------------------------------------------
#  Determines the coefficients a[i], i=1,..,npar of a multilinear model
#     F(x;a) = a[1] * func[1](x) + ... + a[npar] * func[npar](x)
#  and the associated uncertainties, sigma[i], for a set of observed data
#  points by "Chi-square" regression
#  x[]     - x-coordinates of observed data, i = 1,..,n
#  y[]     - y-coordinates of observed data
#  sigmy[] - standard deviations of observed data
#  n       - number of observed data points
#  iopt    - iopt == 0 - initializes sigmy[i] = 1 (equal weights)
#          - iopt != 0 - uses the received values of sigmy[i]
#  a[]     - parameters of the multilinear model (output)
#  sigma[] - uncertainties associated with the model parameters (output)
#  npar    - number of model parameters
#  chi2    - value of Chi-square merit function for the output parameters
#  Func()  - user function returning for a given argument x the values of the
#            basis functions func[i](x):
#               Func(x, func, npar)
#  Calls: GaussJordan (linsys.py)
#-----------------------------------------------------------------------------
   c = [[0]*(n+1) for i in range(n+1)]
   b = [[0]*2 for i in range(n+1)]
   func = [0]*(npar+1)

   if (iopt == 0):
      for i in range(1,n+1): sigmy[i] = 1e0        # iopt = 0: equall weights

   for i in range(1,npar+1):                                  # initialization
      for j in range(1,npar+1): c[i][j] = 0e0
      b[i][1] = 0e0

   for i in range(1,n+1):                          # generate the system matrices
      yi = y[i]
      Func(x[i],func,npar)                         # evaluate the basis functions
      f = 1e0/(sigmy[i]*sigmy[i])
      for j in range(1,npar+1):
         fj = f * func[j]
         for k in range(1,npar+1): c[j][k] += fj * func[k]
         b[j][1] += fj * yi

   det = GaussJordan(c,b,npar,1)                                # solve the system

   for i in range(1,npar+1):
      a[i] = b[i][1]                                          # model parameters
      sigma[i] = sqrt(c[i][i])                        # parameter uncertainties

   chi2 = 0e0                                      # value of Chi-square function
   for i in range(1,n+1):
      Func(x[i],func,npar)                          # evaluate the model function
      f = 0e0
      for j in range(1,npar+1): f += a[j]*func[j]
      f = (y[i] - f)/sigmy[i]
      chi2 += f*f
   return chi2
```

data points in the arrays x and y, along with the standard deviations of the y-coordinates, in the array sigmy. The number of model parameters is specified by the argument npar. If the control parameter iopt is set to 0, all the components of the array sigmy are initialized internally with 1 and the approximation reduces to the least-squares regression based on the merit function

$$S = \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{m} a_j F_j(x_i) \right]^2 .$$

The function MultiFit returns the adjusted model parameters and the associated uncertainties in the array a and, respectively, sigma. As a synthetic quality indicator for the performed fit, the routine also returns via chi2 the value of the χ^2-merit function calculated with the optimal parameters.

The basis functions $F_j(x)$ composing the model have to be evaluated by a separate user function, which is recognized internally by MultiFit under the generic name Func and whose actual name is conveyed by way of the parameter list. For a given argument x, the user function returns the npar values of the basis functions in the components of the array func.

The system of normal equations (9.52) may be solved, in principle, by any general-purpose method. However, since for the evaluation of the parameter uncertainties also the inverse of the system matrix is required, the most convenient turns out to be the Gauss–Jordan method, discussed in Chapter 7, which enables the simultaneous solution of the system and the in-place inversion of the system matrix. The developed routine GaussJordan solves a matrix equation $\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$, where $\mathbf{A}$ is a square $(n \times n)$ matrix, while $\mathbf{B}$ and $\mathbf{X}$ are $(n \times m)$ matrices, replacing on exit the constant terms $\mathbf{B}$ by the solution $\mathbf{X}$ and the system matrix $\mathbf{A}$ by its inverse $\mathbf{A}^{-1}$. By the specific call to GaussJordan from the routine MultiFit, the array c, containing initially the fitting matrix, returns its inverse (the covariance matrix), whereas the array b of constant terms is replaced by the system's solution (the model parameters). As a technical detail, although in the present case the matrix equation reduces to an ordinary linear system with just a vector of constant terms ($m = 1$), for reasons of compatibility with the arguments of GaussJordan, b has to be declared as a two-dimensional array, with npar rows and a single column.

Figure 9.8 illustrates the method of multilinear regression by a simple example, in which the "observed" data points (depicted with circles) are generated by sampling the function $f(x) = \sin x + \cos x$

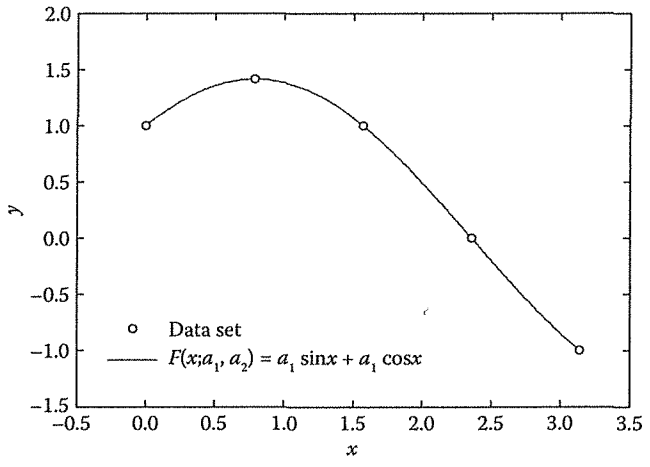

FIGURE 9.8 Multilinear regression performed with the help of the routine MultiFit. The observed data are sampled from the function $f(x) = \sin x + \cos x$.

for the hand-picked arguments 0, $\pi/2$, π, $3\pi/2$, and π. In the sample program from Listing 9.10, the model function is chosen to have two adjustable parameters,

$$F(x; a_1, a_2) = a_1 \sin x + a_2 \cos x,$$

and the evaluation of the basis functions $F_1(x) = \sin x$ and $F_2(x) = \cos x$ is carried out by the user function `Func`. The adjusted model parameters returned by `MultiFit` are $a_1 = 0.999929$ and $a_2 = 1.000212$, with the merit function $\chi^2 = 0.000000$. It is noteworthy that the fitting procedure of the model function recovers within roundoff errors the coefficients of the original function $f(x)$, which has provided the "observed" data.

Generalizing the model function to include more basis functions (1, $\sin x$, $\cos x$, $\sin 2x$, $\cos 2x$, etc.), the routine `MultiFit` can be used, in principle, to extract Fourier series coefficients from tabulated dependences (see Problem 9.9).

In the particular case of the polynomial regression, the model function having m adjustable parameters is a polynomial of degree $(m - 1)$:

$$F(x; \mathbf{a}) = \sum_{j=1}^{m} a_j x^{j-1}. \tag{9.57}$$

Listing 9.10 Application of Multilinear Regression (Python Coding)

```
# Multilinear fit
from math import *
from modfunc import *

def Func(x, func, npar):
   func[1] = sin(x)          # returns the basis functions sin(x) and cos(x)
   func[2] = cos(x)

# main

n = 5                                              # number of observed data
npar = 2                                         # number of model parameters

x     = [0]*(n+1)                              # x-coordinates of observed data
y     = [0]*(n+1)                              # y-coordinates of observed data
sigmy = [0]*(n+1)                       # standard deviations of observed data
func  = [0]*(npar+1)                             # values of basis functions
a     = [0]*(npar+1)                                      # model parameters
sigma = [0]*(npar+1)                           # uncertainties of parameters

x[1] =  0e000; y[1] =  1e000                    # observed data generated from:
x[2] =  0.785; y[2] =  1.414                          # f(x) = sin(x) + cos(x)
x[3] =  1.571; y[3] =  1e000
x[4] =  2.356; y[4] =  0e000
x[5] =  3.141; y[5] = -1e000

iopt = 0                                 # least squares fit: equal errors sigmy
chi2 = MultiFit(x,y,sigmy,n,iopt,a,sigma,npar,Func)

print("Multilinear fit:")
for i in range(1,npar+1):
   print(("a[{0:d}] = {1:8.4f} +/- {2:8.4f}").format(i,a[i],sqrt(sigma[i])))
print("Chi^2 = {0:8.4f}".format(chi2))
```

Listing 9.11 Polynomial Regression (Python Coding)

```
#=============================================================================
def PolFit(x, y, sigmy, n, iopt, a, sigma, npar):
#-----------------------------------------------------------------------------
#  Determines the coefficients a[i], i=1,..,npar of a polynomial model
#     F(x;a) = a[1] * x^(npar-1) + a[2] * x^(npar-2) + ... + a[npar]
#  and the associated uncertainties, sigma[i], for a set of observed data
#  points by "Chi-square" regression
#  x[]     - x-coordinates of observed data, i = 1,..,n
#  y[]     - y-coordinates of observed data
#  sigmy[] - standard deviations of observed data
#  n       - number of observed data points
#  iopt    - iopt == 0 - initializes sigmy[i] = 1 (equal weights)
#          - iopt != 0 - uses the received values of sigmy[i]
#  a[]     - parameters of the polynomial model (output)
#  sigma[] - uncertainties associated with the model parameters (output)
#  npar    - number of model parameters (polynomial order + 1)
#  chi2    - value of Chi-square merit function for the output parameters
#  Calls: GaussJordan (linsys.py)
#-----------------------------------------------------------------------------
   c = [[0]*(n+1) for i in range(n+1)]
   b = [[0]*2 for i in range(n+1)]
   func = [0]*(npar+1)

   if (iopt == 0):
      for i in range(1,n+1): sigmy[i] = 1e0       # iopt = 0: equall weights

   for i in range(1,npar+1):                                  # initialization
      for j in range(1,npar+1): c[i][j] = 0e0
      b[i][1] = 0e0

   for i in range(1,n+1):                        # generate the system matrices
      xi = x[i]; yi = y[i]
      func[npar] = 1e0                          # basis functions 1, x, x^2,...
      for j in range(npar-1,0,-1): func[j] = xi * func[j+1]
      f = 1e0/(sigmy[i]*sigmy[i])
      for j in range(1,npar+1):
         fj = f * func[j]
         for k in range(1,npar+1): c[j][k] += fj * func[k]
         b[j][1] += fj * yi

   det = GaussJordan(c,b,npar,1)                  # solution: corrections da

   for i in range(1,npar+1):
      a[i] = b[i][1]                                        # model parameters
      sigma[i] = sqrt(c[i][i])                    # parameter uncertainties

   chi2 = 0e0                                  # value of Chi-square function
   for i in range(1,n+1):
      xi = x[i]                                 # evaluate the model function
      f = a[1]
      for j in range(2,npar+1): f = f*xi + a[j]
      f = (y[i] - f)/sigmy[i]
      chi2 += f*f

   return chi2
```

The corresponding user function can be coded in the most efficient way by taking advantage of the simple recurrence relation between the successive basis functions 1, x, $x^2, \ldots$:

```
def Func(x, func, npar):   # returns basis functions 1, x, x^2,...,x^(npar-1)
   func[npar] = 1e0                          # basis functions 1, x, x^2,...
   for i in range(npar-1,0,-1): func[i] = x * func[i+1]
```

Yet another option to implement the polynomial regression is to rewrite the routine `MultiFit`, as in Listing 9.11, so as to evaluate the basis functions in-place and not to call any user function. The routine `PolFit` is useful, for example, for extracting power series coefficients from tabulated dependences.

9.7 Nonlinear Regression: The Levenberg–Marquardt Method

The least-squares method can be applied, in principle, for fitting any arbitrary model function to observed data points, but requires, in general, the solution of systems of nonlinear equations for determining the optimal model parameters. The procedure is called in such cases *nonlinear regression* and the corresponding algorithms typically feature a considerable degree of complexity. Unlike the multilinear regression, in which the model parameters are determined in a single step by solving a linear system of equations, the nonlinear regression refines the model parameters iteratively, starting from initial guesses.

The merit function is considered to have the general form:

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - F(x_i; \mathbf{a}) \right]^2 . \tag{9.58}$$

With a view to characterizing the minimum of χ^2, it is useful to consider its series expansion with respect to the parameter values:

$$\chi^2(\mathbf{a} + \delta\mathbf{a}) = \chi^2(\mathbf{a}) + \sum_{j=1}^{m} \frac{\partial \chi^2}{\partial a_j} \delta a_j + \frac{1}{2} \sum_{j,l=1}^{m} \frac{\partial \chi^2}{\partial a_j \partial a_l} \delta a_j \delta a_l + \cdots , \tag{9.59}$$

where $\delta\mathbf{a}$ is the vector of parameter corrections. The merit function shows an extremum for the set of corrections $\delta\mathbf{a}$ which make its gradient vanish, or, equivalently, its partial derivatives with respect to all the parameters:

$$\left. \frac{\partial \chi^2}{\partial a_k} \right|_{\mathbf{a}+\delta\mathbf{a}} = 0, \quad k = 1, 2, \ldots, m. \tag{9.60}$$

The derivatives of χ^2 for the corrected parameters result from the Expansion 9.59,

$$\begin{aligned} \left. \frac{\partial \chi^2}{\partial a_k} \right|_{\mathbf{a}+\delta\mathbf{a}} &= \sum_{j=1}^{m} \frac{\partial \chi^2}{\partial a_j} \delta_{jk} + \frac{1}{2} \sum_{j,l=1}^{m} \frac{\partial \chi^2}{\partial a_j \partial a_l} (\delta_{kj} \delta a_l + \delta a_j \delta_{kl}) + \cdots \\ &= \frac{\partial \chi^2}{\partial a_k} + \sum_{j=1}^{m} \frac{\partial \chi^2}{\partial a_k \partial a_j} \delta a_j + \cdots \end{aligned} \tag{9.61}$$

The first-order derivatives for the reference parameters entering the above expression can be derived from the definition (9.58) of the merit function:

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - F(x_i; \mathbf{a}) \right] \frac{\partial F(x_i; \mathbf{a})}{\partial a_k}. \tag{9.62}$$

Analogously, for the mixed second derivatives, forming the so-called *Hessian matrix* of the merit function, one obtains:

$$\begin{aligned} \frac{\partial \chi^2}{\partial a_k \partial a_j} &= 2 \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left\{ \frac{\partial F(x_i; \mathbf{a})}{\partial a_k} \frac{\partial F(x_i; \mathbf{a})}{\partial a_j} - \left[y_i - F(x_i; \mathbf{a}) \right] \frac{\partial^2 F(x_i; \mathbf{a})}{\partial a_k \partial a_j} \right\} \\ &\approx 2 \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \frac{\partial F(x_i; \mathbf{a})}{\partial a_k} \frac{\partial F(x_i; \mathbf{a})}{\partial a_j}. \end{aligned} \tag{9.63}$$

In the vicinity of the minimum of χ^2, the mixed second derivatives $\partial^2 F(x_i; \mathbf{a}) / \partial a_k \partial a_j$ are negligible in comparison with the first-order ones and, being also weighted with the quantities $[y_i - F(x_i; \mathbf{a})]$, which are randomly signed and distributed, they tend to cancel out statistically by summation. Furthermore, by imposing the vanishing conditions (9.60) on the derivatives of χ^2 and confining their Expansions 9.61 to the linear approximation, there results the following system of linear equations for the parameter corrections (Press et al. 2007):

$$\sum_{j=1}^{m} c_{kj} \delta a_j = b_k, \quad k = 1, 2, \ldots, m, \tag{9.64}$$

with the matrix elements defined by

$$c_{kj} \equiv \frac{1}{2} \frac{\partial \chi^2}{\partial a_k \partial a_j} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \frac{\partial F(x_i; \mathbf{a})}{\partial a_k} \frac{\partial F(x_i; \mathbf{a})}{\partial a_j}, \tag{9.65}$$

$$b_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[y_i - F(x_i; \mathbf{a}) \right] \frac{\partial F(x_i; \mathbf{a})}{\partial a_k}. \tag{9.66}$$

c_{kj} equal half the elements of the Hessian matrix and $\mathbf{C} = [c_{kj}]$ is called *curvature matrix.*

Far from the minimum of χ^2, the Expansion 9.59 may represent a poor local approximation to $\chi^2(\mathbf{a} + \delta\mathbf{a})$. In such a case, in order to accelerate the convergence of the process toward the minimum, one can retain only the diagonal elements of the Hessian matrix and, consequently, the components of the gradient of the merit function (9.61) become

$$\left. \frac{\partial \chi^2}{\partial a_k} \right|_{\mathbf{a}+\delta\mathbf{a}} \approx \frac{\partial \chi^2}{\partial a_k} + \frac{\partial \chi^2}{\partial a_k^2} \delta a_j. \tag{9.67}$$

Making use of the notations (9.65) and (9.66), the conditions of vanishing χ^2 derivatives (9.60) lead to the following simplified equations for the parameter corrections:

$$c_{kk} \delta a_k = b_k, \quad k = 1, 2, \ldots, m. \tag{9.68}$$

The Levenberg–Marquardt method (Marquardt (1963)) has become in practice the standard approach for nonlinear regression. It provides an efficient strategy for determining the optimal corrections of the model parameters by smoothly switching between Equations 9.68, which are applicable far from

the minimum of χ^2, and Equations 9.64, which are applicable close to the minimum. Technically, by introducing a *dimensionless scale factor* λ, the two sets of equations take a unique form:

$$\sum_{j=1}^{m}(1+\lambda\delta_{kj})c_{kj}\delta a_j = b_k, \quad k = 1, 2, \ldots, m, \tag{9.69}$$

where δ_{kj} is the Kronecker delta. For large values of λ, the system matrix is *diagonally dominant* and there results the uncoupled set of equations (9.68), which is applicable far from the minimum. On the contrary, for small values of λ, the system takes the form (9.64) and allows for the parameter corrections to be determined close to the minimum.

Analogously to the multilinear regression, the variance (squared probable imprecision) associated with the adjusted parameter a_j is given by the diagonal element c'_{jj} of the inverse $\mathbf{C}^{-1}$ of the curvature matrix, that is,

$$\sigma^2_{a_j} = c'_{jj}. \tag{9.70}$$

The nondiagonal elements c'_{jk}, in general, represent the *covariances* corresponding to the pairs of model parameters (a_j, a_k).

The corrections to the model parameters have to be calculated iteratively from the linear system (9.69), with the coefficients specified by Equations 9.65 and 9.66, until the maximum relative change of the individual model parameters drops under a predefined tolerance ε:

$$\max_j |\delta a_j/(a_j + \delta a_j)| \leq \varepsilon. \tag{9.71}$$

If there exist vanishing components $a_j + \delta a_j$, rather the absolute corrections have to be employed in expressing this criterion. Iterating up to the machine representation limit is, in general, neither necessary nor useful, since the minimum of the merit function provides at best statistical estimates of the parameters a_j. Another stopping criterion, complementary to the one for the parameter values, checks the degree to which the merit function has converged:

$$|1 - \chi^2(\mathbf{a})/\chi^2(\mathbf{a} + \delta\mathbf{a})| \leq \varepsilon. \tag{9.72}$$

Synthetically, the Levenberg–Marquardt algorithm implies iterating the following steps (Press et al. 2007):

1. Calculate the merit function $\chi^2(\mathbf{a})$ for an initial set of parameters.
2. Consider a relatively small initial value for λ, such as 10^{-3}.
3. Calculate the coefficients c_{kj} and b_k from Equations 9.65 and 9.66, and solve the linear system (9.69) for the parameter corrections $\delta\mathbf{a}$.
4. Evaluate the merit function $\chi^2(\mathbf{a} + \delta\mathbf{a})$ for the corrected parameters.
5. If $\chi^2(\mathbf{a} + \delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, *increase* λ by a factor of 10 and go to 3.
6. If $\chi^2(\mathbf{a} + \delta\mathbf{a}) < \chi^2(\mathbf{a})$, *decrease* λ by a factor of 10 and update $\mathbf{a} + \delta\mathbf{a} \rightarrow \mathbf{a}$.
7. If $\max_j |\delta a_j/a_j| > \varepsilon$ or $|1 - \chi^2(\mathbf{a})/\chi^2(\mathbf{a} + \delta\mathbf{a})| > \varepsilon$ go to 3.

In any attempt to implement the Levenberg–Marquardt method efficiently, one has to evaluate repeatedly the χ^2-merit function and the linearized fitting matrices **C** and **b**, which depend themselves on the derivatives of the model function. To this end, the routine `Deriv` from Listing 9.12 calls the user-defined model function (with the local alias `Func`), calculates by forward finite differences its first derivatives $\partial F(x_i; \mathbf{a})/\partial a_j$ with respect to the parameters a_j, and returns these via the array `dFda` to the calling function `Chi2`. The latter builds from the model function derivatives the fitting matrices **C** and **b** according to Relations 9.65 and 9.66, and returns them together with the χ^2-merit function to the calling routine `MarqFit` (Listing 9.13).