FITTING IN MATLAB

FIT.TEX KB 20030922

KLAUS BETZLER[1], FACHBEREICH PHYSIK, UNIVERSITÄT OSNABRÜCK

This short lecture note presents some aspects of doing fits in MATLAB. Of course it can not cover the wide field of data fitting, it will be mainly restricted to simple problems – erroneous data as a function of one variable.

1 Why Fitting?

There can be different reasons and purposes for fitting data, including

- Getting certain features out of a set of data, e. g. finding a maximum or an inflection point.
- Producing "nice" figures, i. e. plotting curves as *guide for the eye*.
- Describing data by a simpler physical principle, the fit will then yield the parameters in the corresponding physical formula.
- Finding a *lookup formula* for a dependance between different physical properties.

Depending on the purpose, the fit to be used *need* or *need not* have a background in physics. Yet, it always will have some background in mathematics. That means, a fit will always yield some result, will always yield parameters. However, these parameters will not always be physics.

Never mistake *Fitting* for *Physics*.

2 Mathematics

Fitting means to find a mathematical description for data. This mathematical description should be "as close as possible" to the data. Obviously one has to define in a mathematical way what is meant with "as close as possible". Commonly the *root mean squared error* is used as a measure for the deviation[2]. To "fit" then means to find a minimum for the *root mean squared error*.

[1] KLAUS.BETZLER@UOS.DE

[2] As an alternative measure, e. g., the mean absolute value of the error could be used.

To achieve this, a fit function f must be defined which generally contains adjustable parameters, the *fit parameters* $a_i, i = 0 \ldots n$. Let's consider the simple case of a set of data $y_k, k = 1 \ldots M$ which are collected as a function of one independent variable x at the points x_k. We would like the fit function f to describe these data in an approximative way as

$$y_k \approx f(a_0, \ldots a_n, x_k) \; . \tag{1}$$

To find the parameters a_i for the best approximation we have to minimize the appropriately defined deviation. Instead of the *root mean squared error* we can minimize the sum over the squared residuals

$$r = \sum_k (y_k - f(a_0, \ldots a_n, x_k))^2 \tag{2}$$

which will yield the same results.

The minimum is found by forcing the derivatives of r with respect to all a_i to be zero

$$\frac{\partial r}{\partial a_0} = 0, \quad \frac{\partial r}{\partial a_1} = 0, \quad \ldots \; , \quad \frac{\partial r}{\partial a_n} = 0 \; . \tag{3}$$

Whether and how these partial derivatives can be calculated depends on the detailed form of the function f. It is obvious that a fit approximates the data the better the more parameters are used.

Degrees of Freedom: For a fit often a quantity *degrees of freedom* is defined which is the difference between the number of the data points and the number of the parameters

$$df = M - (n + 1) \; . \tag{4}$$

A positive value for df would be wise.

Residuals: The *residuals* are the differences between data values and corresponding calculated fit data.

Norm of Residuals: The *norm* of a mathematical object is a quantity that – taken in the right sense – describes the length, size or extent of this object. Mathematically more precisely, usually the *L2-norm* is meant. Given a vector z

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \tag{5}$$

the L2-norm is defined as

$$||z||_2 = \left(|z_1|^2 + |z_2|^2 + \ldots + |z_n|^2\right)^{1/2} \quad . \tag{6}$$

(For the *Lj-norm* the '2' in all exponents (2 and 1/2) must be replaced by the respective number j. Also commonly used are *L1* and $L\infty$.)

The *norm of residuals* is thus the square root of the r defined in Eq. 2. It is often used as a measure for the *goodness of fit* when comparing different fits.

3 Polynomial Fits

The simplest sort of fit functions are polynomials

$$f(a_0, \ldots a_n, x_k) = a_0 + a_1 x_k + \ldots + a_n x_k^n \; . \tag{7}$$

The sum of the squared residuals is

$$r = \sum_k (y_k - a_0 + a_1 x_k + \ldots + a_n x_k^n)^2 \; , \tag{8}$$

their partial derivative with respect to a_i

$$\frac{\partial r}{\partial a_i} = \sum_k \left[2(y_k - a_0 + a_1 x_k + \ldots + a_n x_k^n) x_k^i\right] \stackrel{!}{=} 0 \quad (x_k^0 = 1) \; . \tag{9}$$

Eq. 9 is a system of $n+1$ linear equations for the $n+1$ variable parameters a_i. It may be solved by standard solution schemes for linear equations. Generally there is one unique solution, no approximative optimization procedures are necessary. Even more – when applied – optimization procedures may lead to erroneous results.

MATLAB

MATLAB: In MATLAB a polynomial fit can be directly performed in the figure window. Click on Tools and Basic Fitting and you can select polynomial orders. The results of the fit – parameters and the norm of the residuals – can be transferred to the workspace for further usage.

For doing it in a functional form, MATLAB provides the function `polyfit`. In the simplest form, you call it for your data vectors x and y through

```
P = polyfit(x,y,n);
```

for the polynomial order n. Except the parameter vector P, `polyfit` can also provide an additional structure S with more information, then you have to call it as

```
[P,S] = polyfit(x,y,n);
```

(more details in the Help).

The corresponding values for the fit curve can be calculate through the function `polyval`. For drawing the fit curve, it is useful to provide a vector X with narrower spacing in order to get a smooth curve

```
X = linspace(min(x),max(x),500);
Y = polyval(P,X);          % values for fit curve
h1 = plot(X,Y,'k',...);    % draw fit curve
hold on;
h2 = plot(x,y,'ko',...);   % draw data
hold off;
```

The function `polyval` can use the additional information S from `polyfit` to provide error estimates (see the Help for more).

4 Parameter-Linear Fits

Similar to polynomial fits are so-called *parameter-linear* fits, i. e. fits to an arbitrary function with the only restriction that this function is linear in the fit parameters. Such a function can be written as a sum of simpler functions

$$f(a_0, \ldots a_n, x_k) = a_0 + a_1 f_1(x_k) + a_2 f_2(x_k) + \ldots + a_n f_n(x_k) \; . \tag{10}$$

Obviously polynomial fits are a special case of parameter-linear fits when you define $f_n(x_k) = x_k^n$.

Again one can calculate the sum of the squared residuals

$$r = \sum_k (y_k - a_0 + a_1 f_1(x_k) + \ldots + a_n f_n(x_k))^2 \; , \tag{11}$$

and their partial derivative with respect to a_i

$$\frac{\partial r}{\partial a_i} = \sum_k \left[2(y_k - a_0 + a_1 f_1(x_k) + \ldots + a_n f_n(x_k)) f_i(x_k) \right] \stackrel{!}{=} 0 \; . \tag{12}$$

And again we have to solve this system of linear equations which now is a little bit more complicated.

MATLAB **MATLAB:** There is no dedicated fit function for this sort of parameter-linear fits in MATLAB. However, MATLAB knows how to solve a system of linear equations. Given such a system of linear equations

$$Az = b \tag{13}$$

where A is the matrix of the coefficients, z is the vector of the variables and b is the right hand side vector of the equations, MATLAB solves this system using the function `mldivide` (*m*atrix *l*eft *divide*). This function can be abbreviated by '\'. The solution vector then is

```
z = mldivide(A,b);
```

or simpler

```
z = A\b;   .
```

As can be seen in Eq. 12, all these wicked products $f_j(x_k)f_i(x_k)$ have to be calculated and summed over k to get the coefficients of the matrix A for MATLAB. This can be written as a matrix product of 2 rectangular-shaped matrices. But MATLAB can help you to avoid all these formal calculations. The functionality of `mldivide` has been extended to non-square matrices A. If A is rectangular, the problem is solved in a "least squares sense", as MATLAB states. That's exactly what one needs for fitting. We have to solve the set of k equations (over-determined system of linear equations for a_i)

$$f(a_0, \ldots, a_n, x_k) = y(k) \tag{14}$$

in a least squares sense for the variables a_i.

Written as matrices and vectors[3]

$$Fa = y \qquad \text{with} \qquad F = \begin{bmatrix} 1 & f_1(x_1) & \cdots & f_n(x_1) \\ 1 & f_1(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & f_1(x_M) & \cdots & f_n(x_M) \end{bmatrix} \tag{15}$$

and solved by MATLAB as

```
a = mldivide(F,y);      or      a = F\y;  .
```

A simple example is shown in Fig. 1, a sinusoidal data set is to be approximated. Suppose you want to know amplitude and phase, the data then are to be fitted by the function

$$y = A \cdot \sin(x + \phi) \tag{16}$$

with the two fit parameters A and ϕ. This fit function can be easily rewritten to the parameter-linear one

$$y = a_1 \sin(x) + a_2 \cos(x) \tag{17}$$

which then is solved.

The data are provided by

[3]The parameter a_0 and the column of 1 s in the matrix F is only included here to show the formal correspondence with the polynomial case. Usually it is omitted.

```
X = linspace(0,20,100);
Y = 1.9*sin(X+1.3*pi);  .
```

Fitting is then a rather short sequence in the script:

```
F1 = sin(X);
F2 = cos(X);
F = [F1',F2'];
a = F\Y';  .
```

Using the fit parameters a, the relevant objects may be calculated

```
Yfit = F*a;
amplitude = norm(a);
phase = atan2(a(2),a(1));
```

the largest part of the script is wasted for plotting and formatting the plot

```
gray = 0.7*[1,1,1];
plot(X,F1,X,F2,'Color',gray,'Linewidth',2);
hold on;
plot(X,Yfit,'k','Linewidth',2);
plot(X,Y,'ko','Linewidth',2,'Markerfacecolor','w');
hold off;
set(gca,'Linewidth',1.5);
set(gca,'Fontunits','normalized','Fontsize',0.06);  .
```

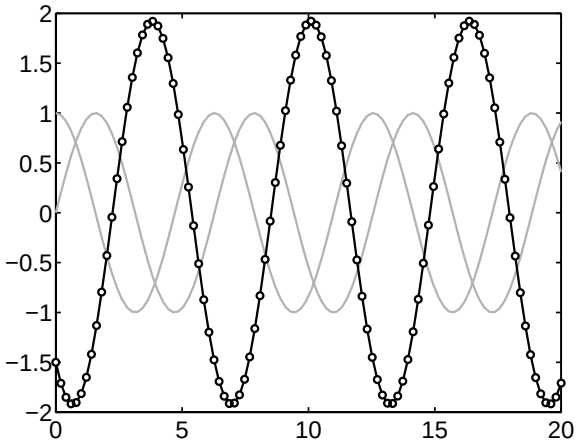

Figure 1: Parameter-linear fit in MATLAB. Dots indicate the original data, gray lines are the two base functions, sine and cosine, and the black line represents the fit result (see text for details).

5 Arbitrary Fit Functions

The two classes of fits described in the preceding sections are mathematically well determined. One can find exact unique solutions for the fit parameters[4]. Therefore one should use polynomial or parameter-linear fits wherever appropriate. Sometimes more complicated fit problems can be transformed to parameter-linear fits. If this is not possible, one has to use arbitrary fit functions which are no longer linear in the fit parameters.

These generalized fit problems have to be solved using optimization algorithms. The function to be optimized (i. e. minimized) is usually the sum of the squared residuals. One thus has to find the minimum of an arbitrary, in general multidimensional function. The variable space of this multidimensional function is defined by the fit parameters. One key issue in all such optimization problems is that there is usually no unique and exact way to find the solution. Approximative algorithms have to be applied. And there is practically no simple way to check whether the algorithm has found the global (absolute) minimum or only a local (relative) one.

MATLAB: To search for the minimum of an arbitrary function, MATLAB provides the function `fminsearch`. To use it, you have to write an additional short function calculating the value of the function to be minimized. If you would want, e. g., to fit a Lorentzian $y = a_1/((x - a_2)^2 + a_3)$ to a data set X_i, Y_i, you should define in MATLAB a function resulting in the sum of the squared residuals

MATLAB

```
function d = devsum(a)
global X Y
d = sum((Y - a(1)./((X-a(2)).^2+a(3))).^2);  .
```

This function is then used in a call to `fminsearch` as the first parameter

```
a3 = ((max(X)-min(X))/10)^2;
a2 = (max(X)+min(X))/2;
a1 = max(Y)*a3;
a0 = [a1,a2,a3];
afinal = fminsearch(@devsum,a0);  .
```

The start values for the parameters (a1, a2, a3) could of course be chosen in a better way when the data set is known. An example for the fit with a Lorentzian is shown in Fig. 2, the data set there was defined by

```
X = linspace(0,100,200);
Y = 20./((X-30).^2+20)+0.08*randn(size(X));  .
```

[4] Problems may occur numerically when the system of linear equations is what you call *badly conditioned*. But that's a known problem in Linear Algebra and we will not deal with it at this point.

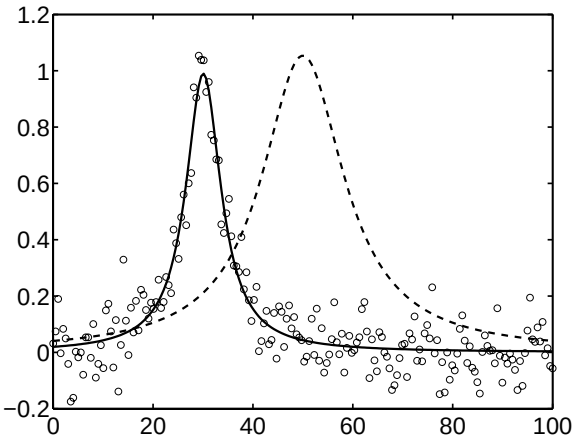

Figure 2: Fitting in MATLAB with `fminsearch`. Dots indicate the original data, dashed line is the starting Lorentzian, full line the fit result (see text for details).

The function `fminsearch` can be tuned by additional options, one can demand a certain accuracy, restrict the number of optimization steps etc. For a detailed description consult Help.

6 Weighted Fit

Up to this point we have assumed that all members of the data set underlying the fit are equal in their reliability. If this is not the case, one should rate the different members with different weights where the weights should be a measure for the reliability. Introducing such weights, not the sum over the squared residuals is to be minimized, instead the sum over the squared *weighted* residuals has to be taken. Instead of Eq. 2 you should use

$$r = \sum_k (w_k(y_k - f(a_0, \ldots a_n, x_k)))^2 \tag{18}$$

where w_k are the weights of the data points k.

MATLAB **MATLAB:** To implement this in MATLAB is straightforward for the parameter-linear and the arbitrary-function fits. You can include it directly in your script. If you want to use the MATLAB function `polyfit`, it has to be adapted by a few additional lines to allow for weighted fits.

You load the function into the MATLAB editor by `edit polyfit.m` and save it to your working directory using a different name (e. g. `wpolyfit`). The vector of the weights w goes as an additional parameter into the parameter list

```
function [p,S,mu] = wpolyfit(x,y,n,w)  .
```

The values of y and the Vandermonde matrix V have to be multiplied by the weights before solving the least squares problem. This is done by some additional lines

```
w = w(:);
y = y.*w;
for j = 1:n+1
  V(:,j) = w.*V(:,j);
end  .
```

Fig. 3 shows the application of such a weighted fit on a set of data with an obvious outlier. The gray curve represents an equally weighted polynomial fit, the black curve a fit where the outlier was given a weight factor of 0.1, all other data a weight of 1.

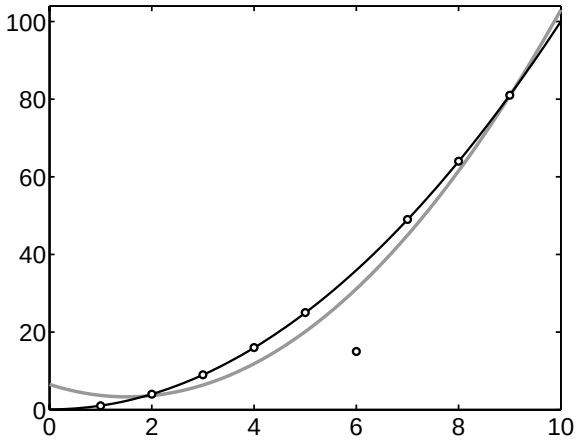

Figure 3: Weighted fit with the accordingly adapted MATLAB function `polyfit`. Dots indicate the original data, gray line is a fit with equal weights, black line the fit with a reduced weight of the outlier (see text for details).

Removal of outliers can be one application of weighted fits, taking into account different experimental errors another one. Given data Y with errors DY, as weights the reciprocal errors can be used (`w = 1./DY` in MATLAB).

Before fitting data you should always analyze what exactly you want to get. An interpolation formula? Features like peak positions in a measured spectrum? The parameters of a physical model? The fit functions you provide will usually depend on this. MATLAB can only deliver the tools for the solution.

Appendix I: M*fit* – A Fitting Environment

Several people have developed a user interface for doing fits in MATLAB. They characterize it as follows:

Mfit is an application for interactive non-linear fitting and data analysis that runs under MATLAB (version 4.2 or later). The primary aim of Mfit is to provide a fast, easy, flexible, and powerful way of fitting arbitrary model functions to two-dimensional (ie. x-y) data.

Easy Use: *Mfit provides an attractive and easy to use point-and-click interface. Most operations (including guessing starting parameters for fits) can be performed via the mouse.*

Easy Data Import: *You can provide your own load routines so that data files in any unusual or proprietary format can be loaded transparently. Load routines for some common formats are supplied.*

Custom Fit Functions: *You can easily write and add your own fitting functions to the library of existing functions. The MATLAB language makes it very easy to write even complex functions, and speed-critical functions can be written in C or FORTRAN.*

Custom Fit Routines: *You can write your own fitting routines, so that you are not tied to any particular fitting algorithm. You need not even use least-squares minimization. You can write interfaces to popular minimization routines such as MINUIT.*

Easy Automation: *Repetitive fitting operations can be automated using a simple batch language.*

Portable: *Mfit is written entirely in the MATLAB language, and so should run on any operating system to which MATLAB has been ported without modification (tested under HP-UX, VMS, Sun OS, and all MS Windows so far.).*

M*fit* is *public domain* software and can be downloaded from

http://www.ill.fr/tas/matlab/doc/mfit4/.

Appendix II: Common Pitfalls

When you are fitting data you should carefully reflect on what you are doing. Many interesting pitfalls are waiting in this field, some typical we should consider here.

A The *Noisy Zero*

In many measurements in physics you measure intensities. And most measurements are noisy. What happens if you have *zero* intensity together with some amount of noise?

We consider the measurement of the exponential decay of a signal (e. g. the time dependance of the luminescence after a pulsed laser excitation of some material). In the left sketch of Fig. 4 the ideal exponential decay signal ($I = A\exp(-t/T)$, $T = 1$ in the x-axis units) is shown (thick line) and some noise is added (thin line). If we try to fit the noisy signal with an exponential function, we get the result shown in the center. The exponential function fits the signal more or less, yet the

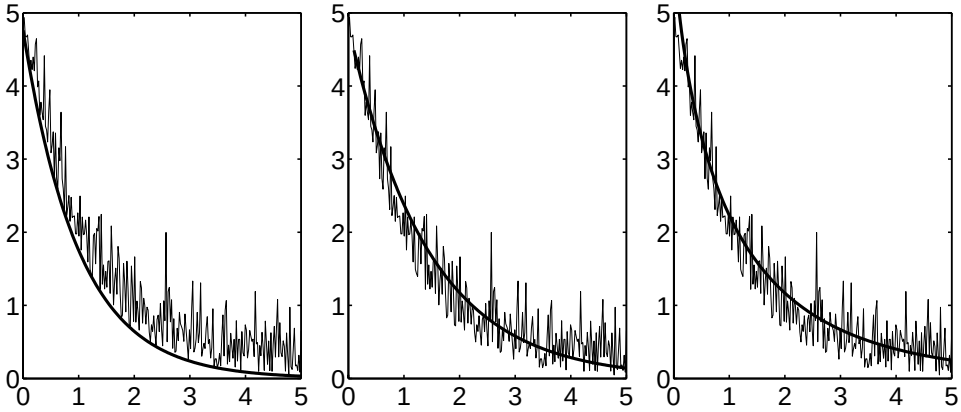

Figure 4: Exponential decay with additional noise. Left: original data, center: exponential fit, right: stretched-exponential fit.

time constant yielded is now $T = 1.42$. To get a better fit, one could try a so-called *stretched-exponential* function (also known as Kohlrausch-Williams-Watts law)

$$I = A\exp(-(t/T)^q) \qquad \text{with} \qquad 0 < q \leq 1\,. \tag{19}$$

This fit is shown on the right, it looks much better than the simple exponential function[5]. The parameters yielded are $T = 1$ and $q = 0.72$.

B Orders of Magnitude

If you measure a quantity over a larger range of magnitudes you have to take care of rating all data in the appropriate way. Let's take as an example characteristic time constants T depending on another material parameter X in a power-function manner

$$T = a \cdot X^p\,. \tag{20}$$

[5]It should be emphasized that in practically all cases, where stretched exponentials are found in physics, they are *not* due to noise but have a clear physical reason.

We set $a = 5$ and $p = -4$ and generate some data using MATLAB

```
X = logspace(0,1.5,5); a = 5; p = -4; T = a*X.^p;  .
```

If we suspect a power function, we can use it as ansatz for the fit function and make the fit with `fminsearch`. As long as the data are exact, we don't run into problems, the left side of Fig. 5 shows the result, both parameters a and p have the expected values. Noisy or outlying data make it more delicate. To test it, on the right subplot one data value is changed – with considerable effect on the exponent p. Nevertheless, the fit seems to be excellent.

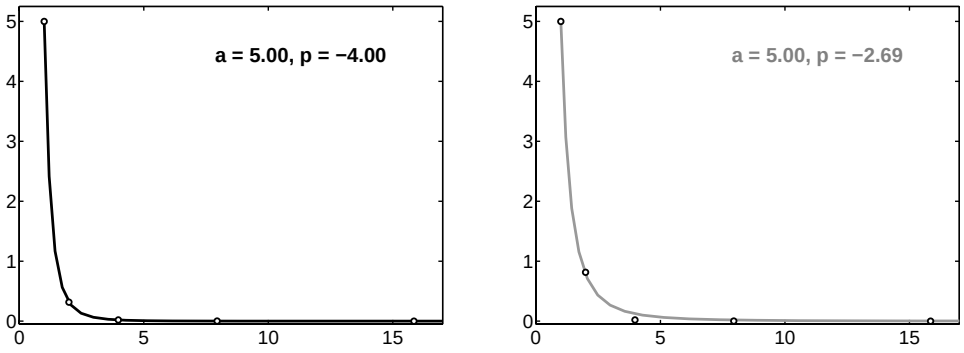

Figure 5: Fitting data with a power function. Left plot: exact data yield exact results, right plot: one outlying data value changes the exponent p considerably.

One way out of the problem is to plot the data in a more appropriate manner, with double logarithmic scaling, and, furthermore, to calculate the fit in this logarithmically scaled space, too. Fig. 6 shows the result, also the fit of the erroneous data yields a reasonable value for the exponent p.

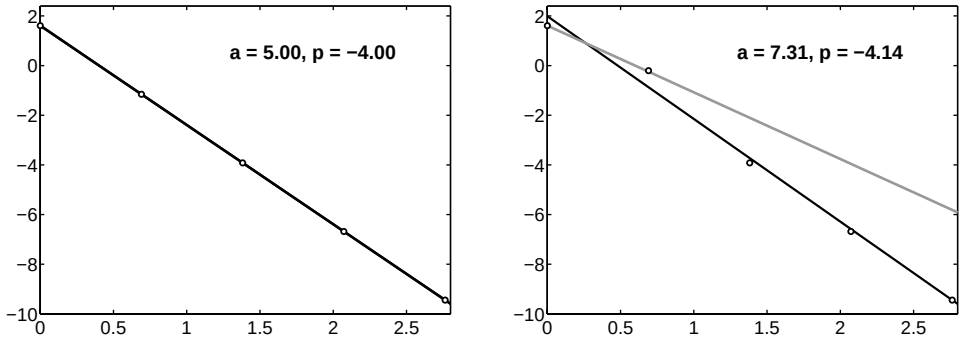

Figure 6: Fitting data with a power function using double logarithmic scaling. Left: exact data, right: one outlying value. For comparison, on the right plot the result of the fit in linear scaling is plotted as gray line.

The comparison with the power function resulting from the fit in linear scaling shows that therein only the first two points are appropriately represented.

C Residual Trends

In a fit usually the L2-norm of the residuals is minimized. Different fit functions applied to a data set may be rated by a comparison of this norm for the respective fits. In most applications this rating is sufficient, especially in cases where the underlying physical description is clear or where only simple approximations are needed (e. g., when a linear or quadratic dependence is searched for).

Complications may occur in cases where you are not sure what fit function to use. *Table lookup*[6] applications are the typical scenarios. In such cases it is useful not only to rate the norm of the residuals but also to plot the residuals to get a graphical impression of typical trends.

We look at an example. In unknown mixtures of two compounds the content of each is to be measured with high accuracy. The two compounds are represented by the respective count rates, N_1 and N_2, in two distinct spectral lines of an X-ray fluorescence spectrum. To get rid of geometry effects, fluctuations in the exciting radiation etc., a normalized relative intensity ratio I_r is calculated

$$I_r = \frac{N_1}{N_1 + N_2} \tag{21}$$

which is a measure for the content of compound 1 in the mixture, c_1. Unfortunately there is no linear dependance between c_1 and I_r.

To get a *lookup table* for the measurement of the unknown samples, known standard mixtures are measured with high accuracy. It is assumed for the first that the deviation from the linear dependance between c_1 and R can be referred to different quantum efficiencies for the two spectral lines. This would yield

$$I_r = \frac{q_1 c_1}{q_1 c_1 + q_2 c_2} = \frac{c_1}{c_1 + Q c_2} \tag{22}$$

with the quantum efficiencies q_i and their ratio Q (for $Q = 1$ we would have linearity as $c_1 + c_2 = 1$). Fig. 7 shows on the left the intensity ratios I_r as a function of the amount of compound 1, c_1, for the standards measured (dots) together with the fit according to Eq. 22 with Q as fit parameter (gray line).

Obviously an excellent fit we could be content with. However, we always want to know, how good we really are. Therefore we plot the residuals (right of Fig. 7) – and – see that we haven't got the whole truth.

How to deal with ? It's table lookup *and* it's physics. So we can choose: improve the table lookup algorithm or improve the physics.

[6] *Table lookup* is quite often used in physics to relate properties depending on each other with no or no simple functional relation. Think of measuring temperature using a thermocouple: You measure the voltage and look into a table where the voltages are referred to temperatures. Usually you don't find the exact voltage you have measured. If your table is accurate enough (as usual for a thermocouple), you can interpolate to get the temperature. If the table data are noisy, you should apply a fitting strategy.

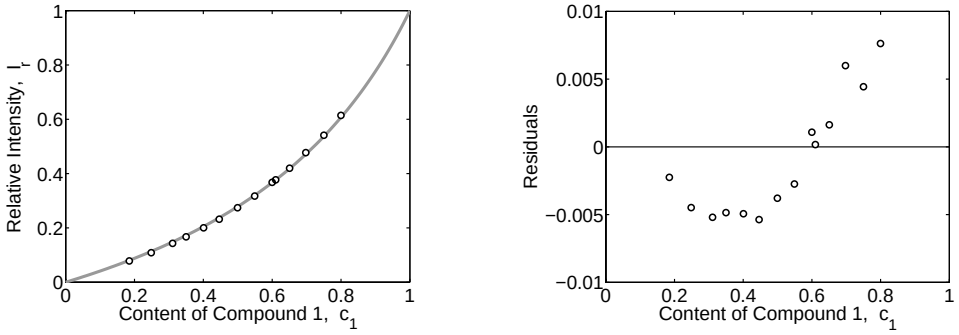

Figure 7: Simple fit according to Eq. 22. Left: measured data and fit curve, right: residuals.

To improve the table lookup is simple and straightforward, we introduce a *correction*. As we can guess from the graph, a quadratic one would be good. The result of a second-order polynomial fit to the residuals is shown in Fig. 8 (left picture). In general, a simple *global* polynomial approach is not obvious, then you can use piecewise (*local*) polynomial fits. In the right picture of Fig. 8 this is shown for the example of piecewise linear fits to the residuals. Of course it looks more noisy than the nice parabola on the left, however, the deviations are small compared to the errors of the measurement. Thus this approach would be acceptable. In most cases such a piecewise fit is better than a nice looking higher order polynomial.

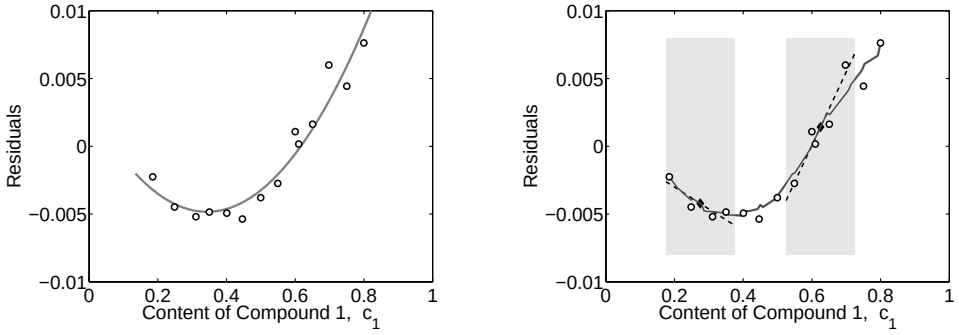

Figure 8: Left: Fit to the residuals using a second-order polynomial. Right: Piecewise linear fit to the residuals. Two fit examples are shown. The areas where the data are taken for the respective linear fits are shaded in gray, the fit lines are dashed. The resulting fit points are shown black-filled. The dark gray piecewise linear curve is the complete result of the fit procedure.

The better way is to improve the physical model on which the fit is based. If possible, one should try to do it. If not the fit then at least your insight into the physics behind it will be improved. In our example we were lucky enough to improve both.

After including things like absorption and reabsorption of the X-rays and dead time of the detector into the model we arrived at the result shown in Fig. 9. The fit curve does not differ much from that of the simple model, yet there is no longer any trend visible with the residuals. Also shown in the right picture are the error margins (standard deviations) $\pm\sigma$ and $\pm 2\sigma$ of the data which help to rate the reliability of the measurements (if more than 68 % of the data are within the $\pm\sigma$-region, there might be something wrong).

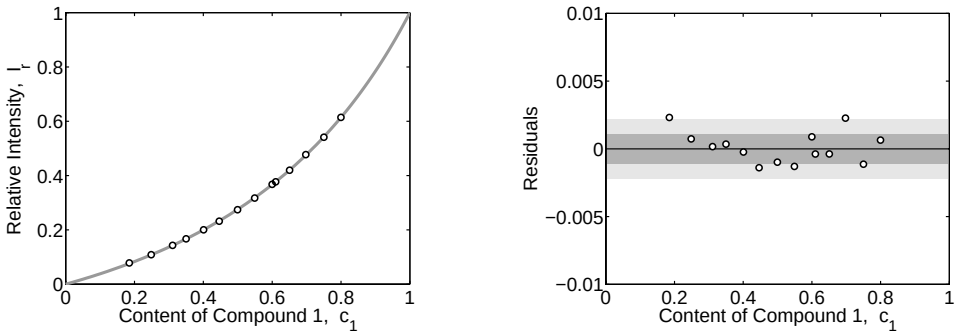

Figure 9: Fit with the improved physical model. Left: data and fit curve, right: residuals. The gray-shaded areas in the right picture are the corresponding $\pm\sigma$- and $\pm 2\sigma$-regions.

D Extrapolation

To rate the reliability of a fit, it is useful to have some data around. But what if you need values outside the range measured data? You can try to *extrapolate*. That might be unproblematic as long as there is some valid physical concept around. Think of a spring in its elastic range. With a force F of 5 Newtons you measure an elongation x of 1 cm, with 10 Newtons then 2 cm, with 15 Newtons 3 cm. Obviously, as you would have expected, a linear dependence of the form

$$x = c \cdot F \quad \text{with} \quad c = 0.002\ \mathrm{mN}^{-1}\ . \tag{23}$$

So you could dare to extrapolate to 20 Newtons. But what if the linear – elastic – range ends at an elongation of 3.5 cm?

Even more difficulties arise when extrapolating data with unknown underlying model. Fig. 10 gives an example. Data are measured in the range $0 \ldots 4$ and should be fitted by some function. The left picture shows a fit with a fourth-order polynomial, the right picture with a hyperbolic tangent. In the range $0 \ldots 4$ both fits are equally good. Thus it's not possible to decide which is the right one knowing only the data of this region. An extrapolation beyond this range, however, yields drastically different values for each of the two fit functions. The gray dots show one possibility how the data could behave beyond the measured range. Would you know this, it would be clear for you which fit is the *right* one.

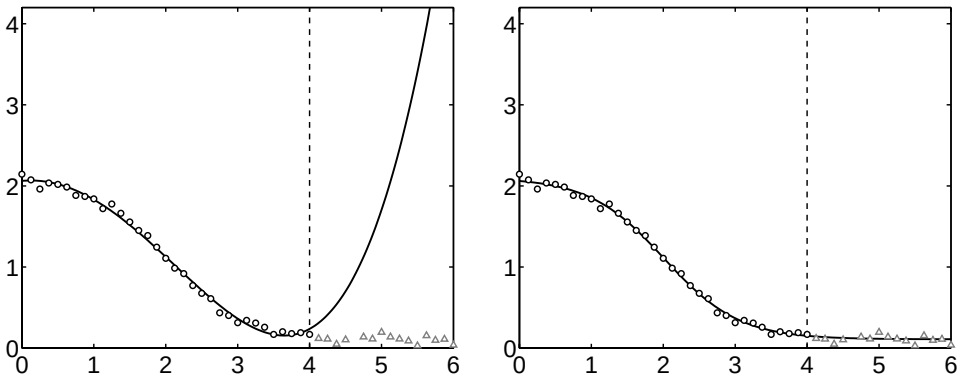

Figure 10: Extrapolation of a fit. The data (black circles) in the region 0...4 (up to the dashed border line) are fitted with different functions, a fourth-order polynomial (left) and a hyperbolic tangent (right). Possible data beyond this range are plotted in gray.

E Fit, Interpolation, or Smoothing ?

When fitting data, you should make sure that it's really that what you want to do. Most table lookup problems, e. g., are better solved by using appropriate *interpolation* schemes. To get nicer-looking curves, *smoothing* algorithms can help you. One of the algorithms used for smoothing, the Savitsky-Golay filter, implements a local, i. e. piecewise, polynomial fit. Doing that, it can also calculate local derivatives of the data. In both – interpolation and smoothing – again MATLAB can do a lot of the work for you. In most cases interpolation and smoothing of data is faster than fitting. And in many cases you get higher accuracy by interpolation or smoothing algorithms as they are working in a restricted local data range which you can size freely.