

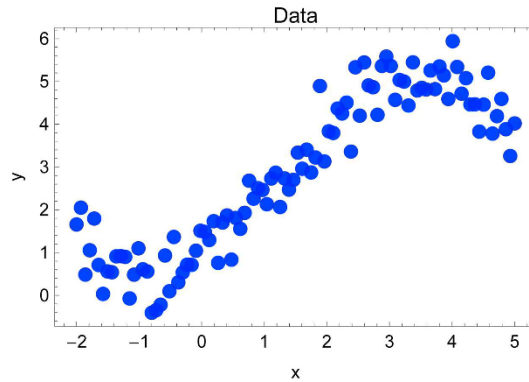
Chapter 2

Curve Fitting

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

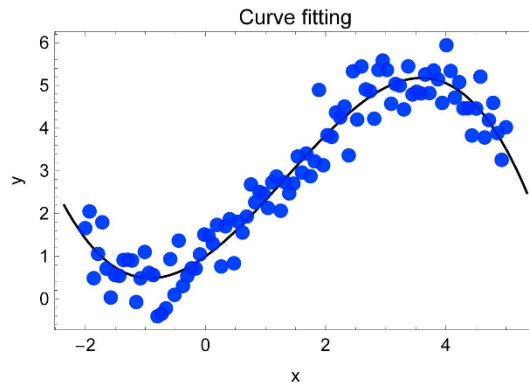
Two-dimensional curve fitting starts with experimental xy-error data (points in diagram below) which consist of data triples (x_i, y_i, σ_i) with an argument value x_i , a corresponding dependent value y_i and the (not illustrated) statistical error σ_i of the y_i value (again note that xy-error data are generated by experimental setups which specify a x_i value and measure a corresponding y_i value for that fixed x_i value where the errors of all x_i values are not taken into account, i.e. all x_i values are considered to be error-free since their errors propagate to corresponding bigger errors σ_i of the dependent y_i values):

```
pureModelFunction=Function[x,1.0+1.0*x+0.4*x^2-0.1*x^3];
argumentRange={-2.0,5.0};
numberOfData=100;
standardDeviationRange={0.5,0.5};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureModelFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data"};
CIP`Graphics`PlotXyErrorData[xyErrorData,labels]
```



Curve fitting tries to adjust a smooth and balancing model function $f(x)$ (solid line in diagram below)

```
modelFunction=a1+a2*x+a3*x^2-a4*x^3;
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3,a4};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,modelFunction,
argumentOfModelFunction,parametersOfModelFunction];
labels={"x","y","Curve fitting"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot"},xyErrorData,curveFitInfo,
CurveFitOptionLabels -> labels];
```



that describes the data adequately (all details will be outlined in a minute). In more mathematical terms curve fitting is a data analysis procedure which tries to construct a linear or non-linear model function

$$y = f(x)$$

from experimental xy-error data. Besides the rare case that the model function $f(x)$ is completely known (then there is nothing to be fitted: The quantity of interest may be directly calculated in this holy grail situation) three different scenarios may be distinguished:

- **Scenario 1:** The structural form of the model function $f(x)$ is theoretically or empirically known but not the values of its parameters, e.g. the structural form is known to be a straight line but the values of its parameters (i.e. of slope and intercept) are unknown.
- **Scenario 2:** The structural form of the model function $f(x)$ is unknown but it may be somehow guessed.
- **Scenario 3:** The structural form of the model function $f(x)$ is unknown and there is no idea what it is about.

Scenario 1 demands a how-to procedure to estimate the unknown parameters of the structurally known model function in an optimum way whereas scenario 2 needs a construction strategy that combines trial and error as well as good guesses in addition (in two dimensions a good guess is quite often feasible in contrast to higher dimensional machine learning problems). For scenario 3 at least some criteria may be derived that allow the construction of something that is smooth and balancing. For scenario 1 (and scenario 2 after the good guess) the estimation of optimum values for the unknown parameters of the model function is the essential step to achieve a good fit. If the statistical distribution of the experimental xy-error data is known this may be performed on a solely statistical ground which then defines the criterion of optimization (see [Hamilton 1964], [Barlow 1989], [Bevington 2002] or [Brandt 2002]). For all further discussions a Gaussian (normal) distribution of experimental errors is always assumed which is the most common case in practice (thanks to the central limit theorem). In addition each data triple (x_i, y_i, σ_i) of the xy-error data is assumed to be statistically independent of each other, i.e. the values of a data triple are by no means influenced by the values of other data triples (which leads to a so called maximum likelihood estimation). Note that this latter assumption is a serious and hard to achieve precondition since a lot of natural (and social) phenomena are subtly correlated to each other. So special care has to be taken for experimental setups to achieve true independence.

If the model function could be successfully fitted to the data it may be used twofold: For interpolation purposes to calculate function values within the experimental argument range $[x_{\min}, x_{\max}]$ as well as for extrapolation purposes to calculate function values outside this argument range. The latter is possible since the structural form of the model function is a priori known. This is a clear difference to mere data smoothing or machine learning methods that have no initial idea about the model function: Their constructed model functions can not be used for extrapolation purposes in principle (multiple linear regression will be an exception but this method is usually not accounted to fall into the machine learning reign).

When a model function is to be guessed (scenario 2) some general considerations should be taken into account. First the number of parameters should be considerably smaller than the number of data (of course this should apply to scenario 1 too): Oth-

erwise a simple look-up table would be easier to create. The number of parameters should be as small as possible or in other words: The model function with fewer parameters that describes the data satisfactorily is preferred to the model function with more parameters. This is a well-known utilization of Occam's razor - one of the philosophical principles of scientific practice that states that the explanation of any phenomenon should make as few assumptions as possible.

In the case that there is no idea of the functional form of a model function (scenario 3) a convincing data smoothing procedure is outlined that uses smoothing cubic splines. It should be clear that this smoothing model function can not be used for extrapolation purposes as mentioned before.

Chapter 2 starts with an outline of necessary basics: The criteria and quantities for curve fitting and data smoothing are intuitively derived with arguments of plausibility only (section 2.1). To tackle scenario 1 quantities and diagrams to assess the goodness of fit are illustrated by means of a perfect straight line fit to simulated data (section 2.2). The empirical construction of a model function for real experimental data on the basis of trial and error in combination with educated guesses is outlined to illustrate scenario 2 as a next step. The extrapolation problem is addressed in particular (section 2.3). Problems and pitfalls of curve fitting tasks are discussed in detail afterwards: They are at heart of this chapter since they are often the hurdle that prevents practitioners from successful data analysis. Fitting non-linear model functions requires adequate start values for all parameters that allow the fitting procedure to succeed: Problems and search strategies are sketched. The extraction of a model function from experimental data may be challenging up to ambiguous which is discussed for difficult curve fitting problems. Model functions themselves may be inappropriately constructed that may lead to fatal pitfalls. A more subtle kind of inappropriateness of a model function is exemplified by an effort to extract information from data that they simply do not contain (section 2.4). The estimation of parameters' errors, possible corrections and the influence of confidence levels are demonstrated afterwards. Parameters' errors are influenced by the precision of data as well as their number: An iterative method for the estimation of the necessary number of data to achieve a desired parameters' precision is suggested. Experimental data of relatively low precision may lead to large parameter errors for specific model functions: This prevents support or rejection of underlying theoretical considerations. In this context there is a strong temptation for educated cheating which means putting up unjustified statements that seem to be advised by the data analysis procedure - an illustrative example is shown. The discussion of the influence of experimental errors on the fitted optimum parameters' values and the related possible problems of data transformations complement this topic (section 2.5). It is often necessary to enhance theoretical model functions by empirical parameters to successfully describe experimental data. An example is discussed that also makes use of the dangerous removal of outliers (section 2.6). Mere data smoothing without any knowledge of a model function (scenario 3) is demonstrated to create a smooth and balancing description of data (section 2.7). Finally the whole chapter is summarized with a few cookbook recipes for successful curve fitting and data smoothing (section 2.8).

2.1 Basics

A curve fitting procedure may be derived with mathematical statistics (see [Hamilton 1964], [Barlow 1989], [Bevington 2002], [Brandt 2002] and [Press 2007]) whereas this section follows an intuitive approach that only uses arguments of plausibility - but of course comes to the same results: How is a model function to be fit? How may data satisfactorily be smoothed?

2.1.1 Fitting data

At first sight it is obvious that a good fit should minimize the so called residuals, i.e. the deviations between experimental values y_i and their corresponding calculated function values $f(x_i)$:

$$y_i - f(x_i) \longrightarrow \text{minimize!}$$

Since positive and negative residuals should be treated equally they may be squared to get rid of the sign:

$$(y_i - f(x_i))^2 \longrightarrow \text{minimize!}$$

The absolute value or a higher even power of the residuals could be taken as well with respect to plausibility but this would lead to other statistics so the square is taken for statistically independent and normally distributed deviations (behind the scenes: The square stems from the square in the exponential term of a normal distribution where the minimum postulation leads to maximum likelihood). The sum of squared residuals of all K xy-error data triples

$$\sum_{i=1}^K (y_i - f(x_i))^2 \longrightarrow \text{minimize!}$$

may be calculated as a quantity to be minimized for a good fit. But so far the errors σ_i of the experimental values y_i are neglected. The smaller a single error σ_i the more precise its corresponding experimental value y_i . If each residual is divided by its corresponding error an individual weight is attributed: The resulting fraction

$$\frac{y_i - f(x_i)}{\sigma_i}$$

is the bigger the smaller the error σ_i is. With the weighted sum of squares

$$\sum_{i=1}^K \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 \longrightarrow \text{minimize!}$$

a plausible minimization quantity is finally achieved: It becomes smaller the smaller the residuals are, i.e. the better the model function $f(x)$ describes the data. Each single residual is weighted with its error σ_i : The smaller an error σ_i the more the corresponding residual $(y_i - f(x_i))$ is taken into account (i.e. the more it contributes to the sum). This minimization process is known in statistics as the method of least squares and the weighted sum of squares is called χ^2 ("chi-square"):

$$\chi^2 = \sum_{i=1}^K \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

If the L parameters a_1 to a_L of the model function f are explicitly written

$$\chi^2(a_1, \dots, a_L) = \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\sigma_i} \right)^2$$

it becomes obvious that the quantity χ^2 is a function of these parameters: The parameters a_1 to a_L of the model function f are the variables of the quantity χ^2 which is to be minimized. Thus minimization of $\chi^2(a_1, \dots, a_L)$ means finding values for the parameters a_1 to a_L so that the value of $\chi^2(a_1, \dots, a_L)$ becomes a global minimum in parameters' value regions that have scientific meaning. The values of the parameters a_1 to a_L at the global minimum of $\chi^2(a_1, \dots, a_L)$ are then called the optimum estimates for the true parameter values in a statistical sense. Note that the functional form of f is assumed to be true as a precondition of all statistical procedures: Only parameter values can be estimated but not the structural form of the function f itself.

In summary a linear or non-linear curve fitting procedure is a mere global minimization of the quantity $\chi^2(a_1, \dots, a_L)$. The global minimum may be calculated analytically in the case that the model function f is linear in its parameters: Then $\chi^2(a_1, \dots, a_L)$ is a parabolic hyper surface and possesses exactly one minimum. But it may only be approximated with an iterative search strategy in the case that f is non-linear in its parameters (compare chapter 1 and [FitModelFunction] in the references). In the latter case the quantity $\chi^2(a_1, \dots, a_L)$ may contain multiple minima and the minimization procedure may fail (e.g. get stuck in a local minimum, exceed the defined maximum number of iterations etc.). Failure will be explicitly explored and discussed in subsequent sections.

2.1.2 Useful quantities

There are a number of related statistical quantities that will prove to be useful for further discussions. If the model function describes the data well the residuals $(y_i - f(x_i))$ should be comparable in size to the errors σ_i on average (otherwise the errors σ_i would not be true errors but systematically too large or too small on average). This means that the fractions

$$\frac{y_i - f(x_i)}{\sigma_i} \approx 1$$

should be close to 1 on average. So the sum of squares evaluates approximately to

$$\chi^2 = \sum_{i=1}^K \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 \approx \sum_{i=1}^K (1)^2 = \sum_{i=1}^K 1 = 1 + 1 + \dots + 1 = K$$

With this result in mind a statistical quantity named χ_{red}^2 ("reduced chi-square") can be defined as

$$\chi_{\text{red}}^2(a_1, \dots, a_L) = \frac{\chi^2(a_1, \dots, a_L)}{K-L} = \frac{1}{K-L} \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\sigma_i} \right)^2 \approx 1 \text{ for } K \gg L$$

which evaluates to a value close to 1 for a good fit since the number of data K should be considerably larger than the number of parameters of the model function L , i.e. $K \gg L$. The denominator $(K - L)$ is called degrees of freedom since the parameter values are deduced from the data. The residuals of a fit may be condensed into the single statistical quantity σ_{fit} called the standard deviation of the fit. If all errors σ_i are identical (i.e. equal to σ) the standard deviation of the fit is defined as

$$\sigma_{\text{fit}} = \sqrt{\frac{1}{K-L} \sum_{i=1}^K (y_i - f(x_i, a_1, \dots, a_L))^2} \text{ for } \sigma_i = \sigma ; i = 1, \dots, K$$

In general with individual errors σ_i the standard deviation of the fit is expressed as

$$\sigma_{\text{fit}} = \sqrt{\frac{1}{K-L} \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\sigma_i} \right)^2} / \sqrt{\frac{1}{K} \sum_{i=1}^K \frac{1}{\sigma_i^2}}$$

where the latter equation reduces to the previous one in the case of equal σ_i . The statistical standard deviation of the fit is similar to a purely empirical quantity called the **root mean squared error (RMSE)**. In this context the RMSE of a fit is simply defined as

$$\text{RMSE} = \sqrt{\frac{1}{K} \sum_{i=1}^K (y_i - f(x_i))^2}$$

A RMSE may readily be generalized to machine learning applications for problems in multiple dimensions. The quantities $\chi_{\text{red}}^2(a_1, \dots, a_L)$, σ_{fit} and RMSE respectively may be used to assess the goodness of a fit. As far as the data's errors are concerned a situation quite often encountered in practice is the following: The precise statistical errors σ_i of the y_i values are unknown, but weights w_i for the y_i values are available. The relation of the weights w_i and their corresponding statistical errors σ_i can be written as

$$\sigma_i = \frac{\alpha}{w_i}$$

where the factor α is used to calculate a statistical error from its corresponding weight. Weights are defined to be the heavier the bigger they are: Statistical errors lead to higher weights the smaller they are. Therefore weights and errors are inversely proportional by the constant factor α . If only weights are known the factor α is unknown. But a reasonable estimate of α may be obtained from the χ^2_{red} value (which should be close to 1 as mentioned before):

$$\begin{aligned}\chi^2_{\text{red}}(a_1, \dots, a_L) &= \frac{\chi^2(a_1, \dots, a_L)}{K-L} = \frac{1}{K-L} \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\sigma_i} \right)^2 \approx 1 \\ \frac{1}{K-L} \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\frac{\alpha}{w_i}} \right)^2 &= \frac{1}{\alpha^2(K-L)} \sum_{i=1}^K w_i^2 (y_i - f(x_i, a_1, \dots, a_L))^2 \approx 1 \\ \alpha &\approx \sqrt{\frac{1}{(K-L)} \sum_{i=1}^K w_i^2 (y_i - f(x_i, a_1, \dots, a_L))^2}\end{aligned}$$

In practice it is common to correct the errors σ_i of the xy-error data with this method: The original errors $\sigma_{\text{original},i}$ of the xy-error data are assumed to be weights only

$$w_i = \frac{1}{\sigma_{\text{original},i}}$$

and are transformed after the fit into the corrected errors $\sigma_{\text{corrected},i}$:

$$\sigma_{\text{corrected},i} = \frac{\alpha}{w_i} = \frac{\alpha}{\frac{1}{\sigma_{\text{original},i}}} = \alpha \sigma_{\text{original},i}$$

These corrected errors $\sigma_{\text{corrected},i}$ are then used for the derivation of further statistical quantities related to the fit - above all the estimation of errors $\sigma_{a_1}, \dots, \sigma_{a_L}$ of the model function's parameters a_1, \dots, a_L .

2.1.3 Smoothing data

When it comes to mere data smoothing statistics is no longer helpful. As already pointed out statistics is not able to guess a model function in principal - it may only estimate optimum values of a structurally known model function's parameters and related quantities with a bunch of statistical preconditions (like independent and normally distributed data). Therefore data smoothing comprises a set of techniques that somehow construct a smooth and balancing interpolating model function from experimental xy-error data (extrapolation is of course not possible). There is no objective way to smooth data so there is nothing like the best smoothing technique. With data smoothing we are back to the jungle where everything is allowed that leads to a satisfactory result. Among the numerous techniques for smoothing experimental

xy-error data the smoothing cubic splines method is sketched in the following (see [Reinsch 1967] and [Reinsch 1971]). This method seems to produce satisfactory results accepted by experimental scientists in general - but this technique is by no means better or superior to others. Data smoothing with cubic splines, i.e. cubic polynomials

$$y = f(x) = a_1 + a_2x + a_3x^2 + a_4x^3,$$

uses the already sketched χ_{red}^2 value

$$\chi_{\text{red}}^2 = \frac{\chi^2}{K} = \frac{1}{K} \sum_{i=1}^K \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

as a reasonable first control parameter: Good data smoothing should lead to a smoothing model function with

$$\chi_{\text{red}}^2 \approx 1$$

For convenience the same notation is used for data smoothing as for statistically fitting model functions. But data smoothing is not statistically based. Quantities like χ_{red}^2 have no longer any statistical meaning but are simply used as helpful quantities for the smoothing task. Therefore χ_{red}^2 is set to $\frac{\chi^2}{K}$ since there are no statistical degrees of freedom for data smoothing. The same applies to quantities like σ_{fit} . They also use the number of data K instead of the degrees of freedom within this context. The cubic splines are constructed from data point to data point, i.e. for K data points $(K - 1)$ cubic splines have to be used. These cubic splines must be adjusted to achieve the initially defined χ_{red}^2 value together with the constraint of a criterion of smoothness: The resulting smoothing model function (composed of the piecewise cubic splines) should possess the smallest overall curvature possible to achieve the predefined χ_{red}^2 value. Since the curvature is measured by the second derivative $f''(x)$ of the model function the integral of the square of the second derivative over the argument interval $[x_1, x_L]$ is to be minimized

$$\int_{x_1}^{x_L} \left(\frac{d^2 f(x)}{dx^2} \right)^2 dx \longrightarrow \text{minimize!}$$

where the xy-error data are assumed to be sorted ascending according to their argument values x_i . The square of the second derivative is used for equal treatment of positive and negative curvature. Both criteria are of course contradicting each other: The smaller the χ_{red}^2 value the larger the curvature integral value and vice versa. With this mutual interplay a satisfactory smooth and balancing model function may be constructed.

2.2 Evaluating the goodness of fit

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

A simple example is used to demonstrate the curve fitting procedure and the evaluation of the goodness of fit. One thousand (x_i, y_i, σ_i) data triples

```
numberOfData=1000;
```

are simulated around the straight line $y = f(x) = 1 + 2x$

```
pureOriginalFunction=Function[x,1.0+2.0*x];
```

in the argument range [2, 5]

```
argumentRange={2.0,5.0};
```

with a relative error of 5% of the function value (since the straight line is constantly increasing a minimum argument value of 2.0 leads to a minimum function value of 5.0: A relative error of 5% for 5.0 is an absolute value of 0.25. A maximum argument value of 5.0 corresponds to a function value of 11.0 with a 5% relative error of 0.55)

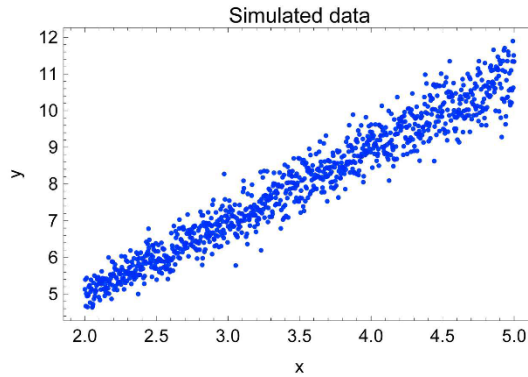
```
errorType="Relative";
standardDeviationRange={0.05,0.05};
```

using the CIP CalculatedData package. All data are normally distributed around their function values, the relative error denotes the standard deviation of the normal distribution used for the data generation (i.e. for a y value of 5 a standard deviation of 0.25 is used, for a y value of 11 a standard deviation of 0.55 respectively):

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange,
CalculatedDataOptionErrorType -> errorType];
```

Here is a plot of the mere simulated data:

```
labels={"x","y","Simulated data"};
pointSize=0.01;
CIP`Graphics`PlotXyErrorData[xyErrorData,labels,
GraphicsOptionPointSize -> pointSize]
```



Curve fitting procedures are performed with the CIP CurveFit package. For a fit the model function itself, the argument and the parameters of the model function must be defined

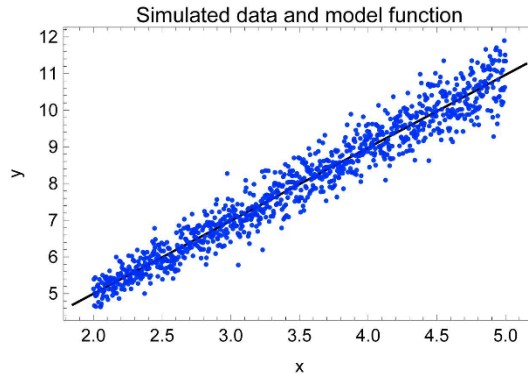
```
modelFunction=a1+a2*x;
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2};
```

and submitted together with the xy-error data to the FitModelFunction method to produce a result captured in a curveFitInfo data structure (see [FitModelFunction] for algorithmic details):

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction];
```

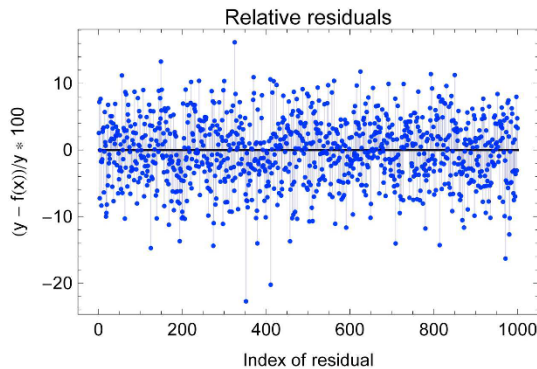
If no error messages are thrown the fit procedure was successful and results can be inspected. The function plot with the fitted straight line and the simulated data painted above provides a first impression:

```
labels={"x","y","Simulated data and model function"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot",xyErrorData,
  curveFitInfo,
  GraphicsOptionPointSize -> pointSize,
  CurveFitOptionLabels -> labels];
```



The fit looks perfect which is also affirmed by inspection of the residuals, i.e. the deviations between the data and the model function: The residuals plot for the relative residuals

```
CIP `CurveFit `ShowFitResult[{"RelativeResidualsPlot"}, xyErrorData,
  curveFitInfo, GraphicsOptionPointSize -> pointSize];
```



exhibits statistically distributed residuals predominantly in the expected value range of $\pm 5\%$ without any systematic deviation patterns. Residuals plots are probably the most important goodness-of-fit visualizations: If they look good the fit in general is good (but compare comments on educated cheating below). Note that the index of an residual corresponds to the x value of its data triple: Residual with index 1 corresponds to the data triple with the smallest x -value, the residual with the highest index to the data triple with the maximum x -value. The standard deviation of the fit σ_{fit}

```
CIP `CurveFit `ShowFitResult[{"SDFit"}, xyErrorData, curveFitInfo];
```

Standard deviation of fit = 3.699×10^{-1}

lies well within the range of (absolute) simulated errors from 0.25 to 0.55. The value of χ^2_{red}

```
CIP `CurveFit `ShowFitResult[{"ReducedChiSquare"}, xyErrorData, curveFitInfo];
```

Reduced chi-square of fit = 9.955×10^{-1}

is close to 1 as expected. The fitted model function is:

```
CIP `CurveFit `ShowFitResult[{"ModelFunction"}, xyErrorData, curveFitInfo];
```

Fitted model function:
 $1.01901 + 1.98941x$

Note that the estimated optimum parameter values are not identical to the true parameter value of 1.0 and 2.0 used for the data generation. The errors of the simulated data are propagated to corresponding errors of the estimated optimum parameter's values so the latter are also not exact but biased by errors:

```
CIP `CurveFit `ShowFitResult[{"ParameterErrors"}, xyErrorData, curveFitInfo];
```

	Value	Standard error	Confidence region
Parameter a1 =	1.01901	0.0454137	{0.973574, 1.06445}
Parameter a2 =	1.98941	0.014094	{1.9753, 2.00351}

The estimated optimum value of parameter a_1 is 1.02, its standard error is 0.05: So the parameter value lies with a standard statistical probability of 68.3% in the confidence region 1.02 ± 0.05 , i.e. interval [0.97, 1.07]. Within linear statistics an awful lot of additional statistical quantities could be deduced. Within the scope of this book the discussion is restricted to basic quantities that play the most important role for evaluation and analysis purposes and those quantities and diagrams that may readily be generalized to machine learning applications for problems with more dimensions. The empirical **root mean squared error** RMSE should also lie within the range of (absolute) simulated errors from 0.25 to 0.55

```
CIP `CurveFit `ShowFitResult[{"RMSE"}, xyErrorData, curveFitInfo];
```

Root mean squared error (RMSE) = 4.044×10^{-1}

and is similar to σ_{fit} as expected. The mean, median, standard deviation and maximum values of the (absolute) relative residuals do correspond perfectly to the simulated errors:

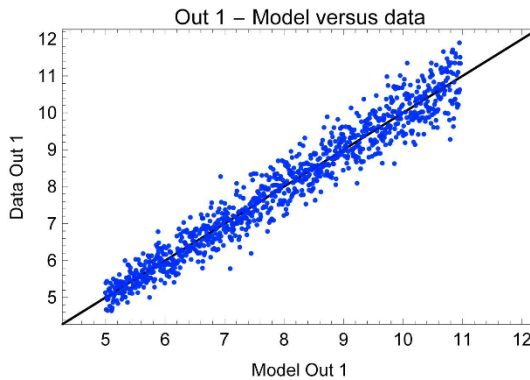
```
CIP 'CurveFit' ShowFitResult[{"RelativeResidualsStatistics"},
  xyErrorData, curveFitInfo];
```

Definition of 'Residual (percent)': $100 \cdot (\text{Data} - \text{Model}) / \text{Data}$

Out 1 : Residual (percent): Mean/Median/Maximum Value = $4.01 / 3.28 / 2.27 \times 10^1$

Out 1 means output component 1: In two-dimensional curve fitting there is only one output component whereas machine learning problems with more dimensions may contain several output components. Another frequently used diagram is the model-versus-data plot: The output (function value) of the model function is plotted against the corresponding data value:

```
CIP 'CurveFit' ShowFitResult[{"ModelVsDataPlot",
  "CorrelationCoefficient"}, xyErrorData, curveFitInfo,
  GraphicsOptionPointSize -> pointSize];
```

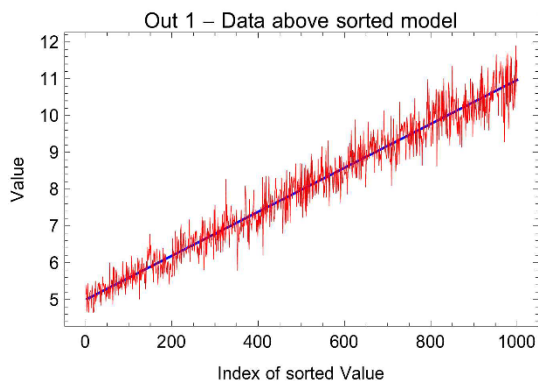


Out 1 : Correlation coefficient = 0.973565

A statistical (Pearson) correlation coefficient was calculated in addition that condenses the agreement between data and output values into a single quantity (where a value closer to one means a desired high correlation between both quantities and a value closer to zero an unwanted low correlation which thus motivates a closer look

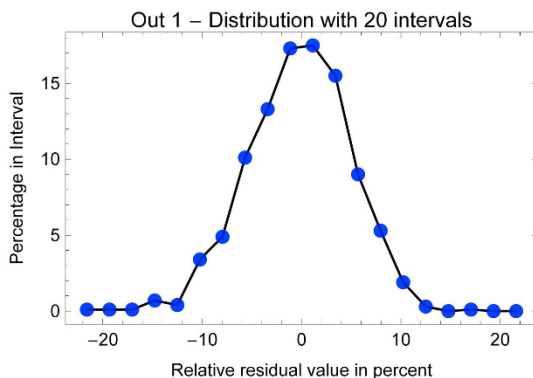
at the used model function with respect to its appropriateness). In an alternative diagram all model function values are sorted in ascending order and are jointly plotted with the corresponding data values above:

```
CIP `CurveFit`ShowFitResult[{"SortedModelVsDataPlot"}, xyErrorData,
  curveFitInfo];
```



The data line above should statistically/randomly crawl around the model line below (the model function outputs) as shown in this perfect example. If the statistical distribution of relative residuals is approximated by the frequency of relative residuals within a number of interval bins (default: 20 bins) that cover the whole range of relative residual values a normal distribution around zero is approximated as expected

```
CIP `CurveFit`ShowFitResult[
  {"RelativeResidualsDistribution"}, xyErrorData,
  curveFitInfo];
```



since a normal distribution was used to generate the data. The width of the approximated Gaussian bell curve corresponds perfectly to the 5% value of the standard deviation used for the data generation above. All plots reveal an excellent and very convincing model function fit. In the next section it is shown how the sketched quantities and diagrams may be utilized to construct a model function for real experimental data.

2.3 How to guess a model function

```
Clear["Global`*"];
<<CIP`ExperimentalData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

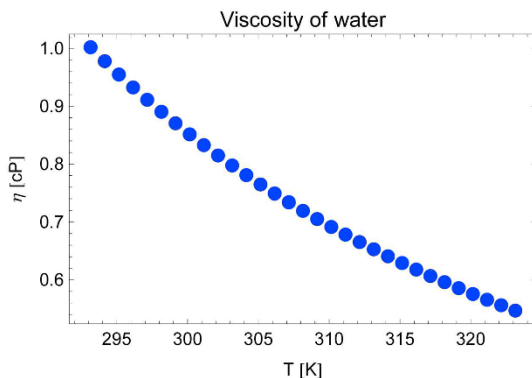
As a practical example a model function for the temperature dependence of the viscosity of water is to be constructed. The viscosity of a liquid is a dynamic property which is the result of the specific molecular interactions describable in the reign of quantum theory. But the dynamics of these interactions is too complex to be calculated ab-initio on the grounds of today's science. Moreover water is not a simple liquid in chemical terms although it is so well-known from everyday life: The water molecules form specific dynamic supramolecular structures due to their ability to create hydrogen bonds - specific weak quantum-mechanical interactions that also hold our DNA strands together. The experimental data are provided by the CIP ExperimentalData package (see Appendix A for reference):

```
xyErrorData=CIP`ExperimentalData`GetWaterViscosityXyErrorData[];
```

They describe the temperature dependence of the viscosity η of water in the temperature range from 293.15 to 323.15 K (20 to 50 degree Celsius) with a very

small estimated experimental error of 0.0001 (10^{-4}) cP (centi-Poise is the scientific unit of viscosity) as is illustrated by the mere data plot:

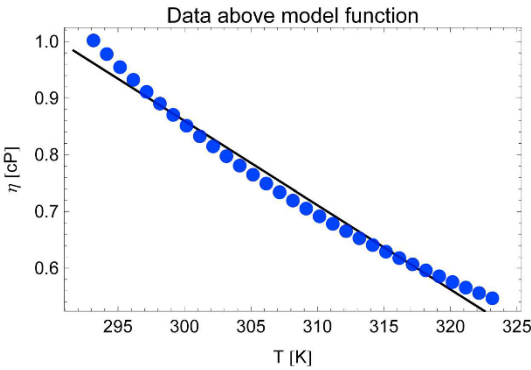
```
labels={"T [K]", "\[Eta] [cP]", "Viscosity of water"};
CIP`Graphics`PlotXyErrorData[xyErrorData, labels]
```



The dependence of the viscosity on the temperature is distinct but not dramatically non-linear as may be shown by an initial straight-line fit:

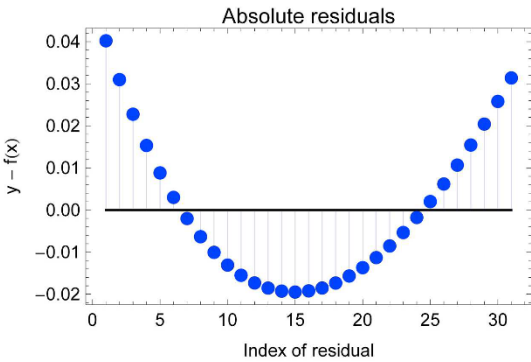
$$\eta = f(T) = a_1 + a_2 T$$

```
modelFunction=a1+a2*T;
argumentOfModelFunction=T;
parametersOfModelFunction={a1,a2};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction];
labels={"T [K]", "\[Eta] [cP]", "Data above model function"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot"},xyErrorData,
  curveFitInfo,CurveFitOptionLabels -> labels];
```



But the residuals (i.e. the deviations between data and linear model) are orders of magnitude larger than the experimental errors and they reveal a distinct systematic deviation pattern:

```
CIP `CurveFit `ShowFitResult[{"AbsoluteResidualsPlot",
"CorrelationCoefficient"},xyErrorData,curveFitInfo];
```



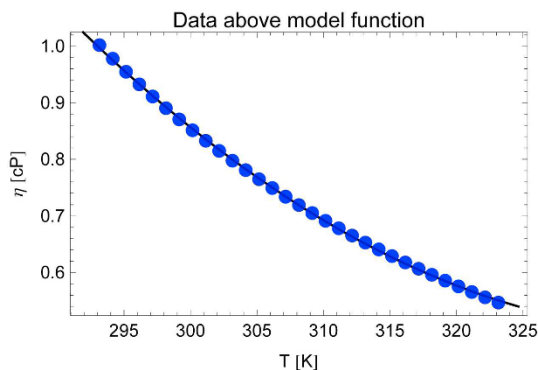
Out 1 : Correlation coefficient = 0.991451

Thus a linear straight line is only a poor model for the data. Note that the popular correlation coefficient is very close to 1 which indicates a high correlation between data and machine output: This is often cited by practitioners as a convincing goodness of fit criterion but it is a number which has to be judged with caution (see discussion below). An improvement may be attempted by introduction of a third parameter to build a (non-linear) quadratic parabola

$$\eta = f(T) = a_1 + a_2T + a_3T^2$$

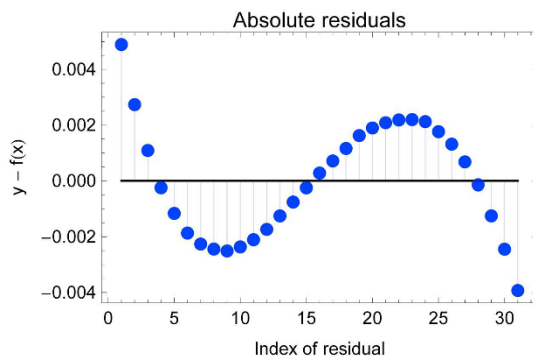
as a model function:

```
modelFunction=a1+a2*T+a3*T^2;
parametersOfModelFunction={a1,a2,a3};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction];
CIP`CurveFit`ShowFitResult[{"FunctionPlot"},xyErrorData,
curveFitInfo,CurveFitOptionLabels -> labels];
```



The function plot looks better and the residuals are reduced by an order of magnitude but are still beyond acceptability:

```
CIP`CurveFit`ShowFitResult[{"AbsoluteResidualsPlot",
"CorrelationCoefficient"},xyErrorData,curveFitInfo];
```

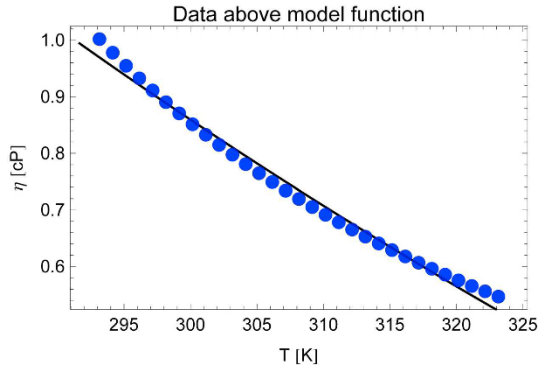


Out 1 : Correlation coefficient = 0.999888

In principal the degree of the fit polynomial could be raised with additional parameters to improve the fit but this strategy is generally a poor one: The high-order polynomials tend to oscillate and may not be predictive for extrapolation or even interpolation purposes (also compare below). Since the viscosity is decreasing with increasing temperature a two-parameter inversely proportional approach seems to be a plausible alternative trial:

$$\eta = f(T) = a_1 + \frac{a_2}{T}$$

```
modelFunction=a1+a2/T;
parametersOfModelFunction={a1,a2};
curveFitInfo=CIP'CurveFit'FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction];
CIP'CurveFit'ShowFitResult[{"FunctionPlot"},xyErrorData,
curveFitInfo,CurveFitOptionLabels -> labels];
```

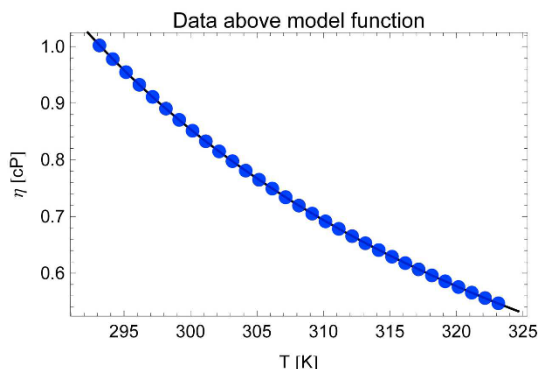


Unfortunately there is no real improvement but it might be a good idea to shift the data along the T axis with a third parameter in addition:

$$\eta = f(T) = a_1 + \frac{a_2}{a_3 - T}$$

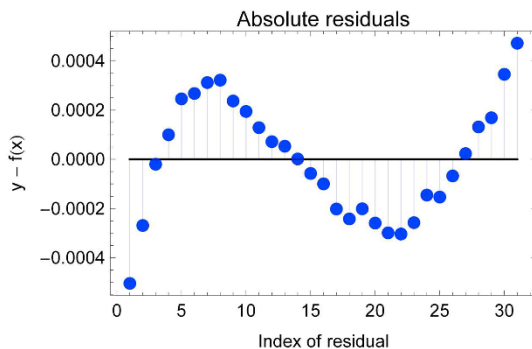
```
modelFunction=a1+a2/(a3-T);
parametersOfModelFunction={a1,a2,a3};
startParameters={a1,1.0},{a2,-10.0},{a3,250.0};
curveFitInfo=CIP'CurveFit'FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP'CurveFit'ShowFitResult[{"FunctionPlot"},xyErrorData,
curveFitInfo,CurveFitOptionLabels -> labels];
```

Note that start parameters had to be introduced to perform a successful fit: This necessity is addressed in the next sections to ease the current discussion.



For the first time the function plot seems to be convincing. The residuals plot shows a dramatic improvement

```
CIP`CurveFit`ShowFitResult[{"AbsoluteResidualsPlot",
  "CorrelationCoefficient"},xyErrorData,curveFitInfo];
```



Out 1 : Correlation coefficient = 0.999998

with residuals in the order of the experimental error. But an unlovely systematic deviation pattern can still be detected: This indicates that the true functional form is still missed. As another alternative a two-parameter decaying exponential function may be tried

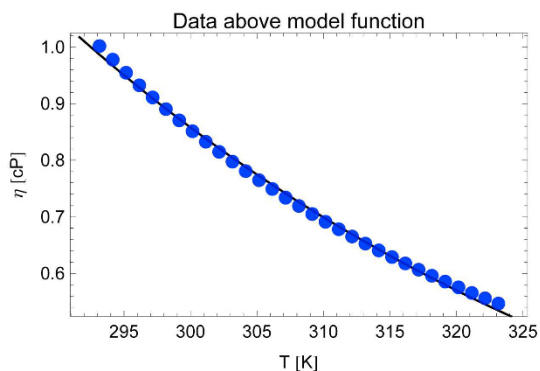
$$\eta = f(T) = a_1 \exp\{a_2 T\}$$

```
modelFunction=a1*Exp[a2*T];
parametersOfModelFunction={a1,a2};
```

```

startParameters={{a1,0.1},{a2,-0.001}};
curveFitInfo=CIP'CurveFit'FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters];
CIP'CurveFit'ShowFitResult[{"FunctionPlot"},xyErrorData,
curveFitInfo,CurveFitOptionLabels->labels];

```

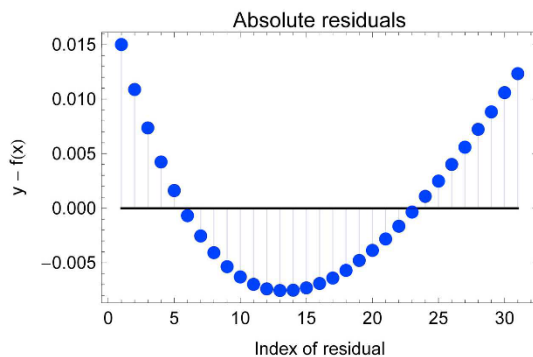


again with a poor result

```

CIP'CurveFit'ShowFitResult[{"AbsoluteResidualsPlot",
"CorrelationCoefficient"},xyErrorData,curveFitInfo];

```

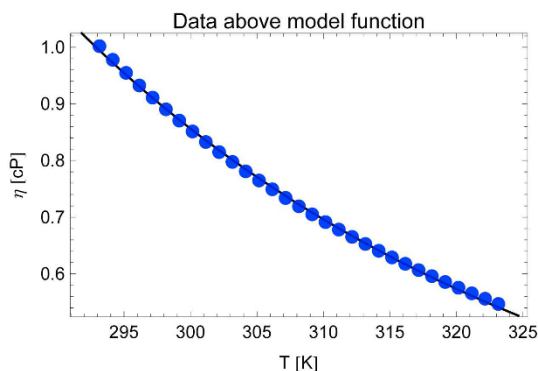


Out 1 : Correlation coefficient = 0.998755

so the use of an inverse argument in the exponential may be a choice

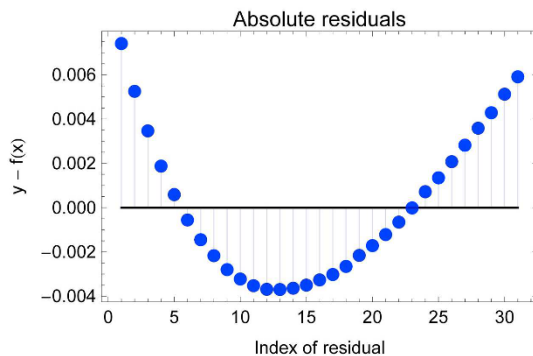
$$\eta = f(T) = a_1 \exp\left\{\frac{a_2}{T}\right\}$$

```
modelFunction=a1*Exp[a2/T];
parametersOfModelFunction={a1,a2};
startParameters={{a1,0.1},{a2,1000.0}};
curveFitInfo=CIP'CurveFit'FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters];
CIP'CurveFit'ShowFitResult[{"FunctionPlot"},xyErrorData,
  curveFitInfo,CurveFitOptionLabels -> labels];
```



that actually offers a better outcome:

```
CIP'CurveFit'ShowFitResult[{"AbsoluteResidualsPlot",
  "CorrelationCoefficient"},xyErrorData,curveFitInfo];
```

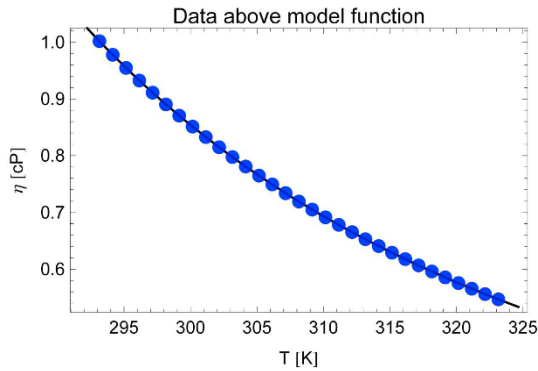


Out 1 : Correlation coefficient = 0.999704

The introduction of the exponential function with a reciprocal argument produced the best two-parameter fit so far (this is also a historical result obtained by Andrade, see [Andrade 1934]: Note that the fitted model function can be linearized by a logarithmic transformation - the only feasible solution for non-linear problems in the precomputing era). Since shifting along the T axis with a third parameter was successful earlier it is tried again with the new functional form:

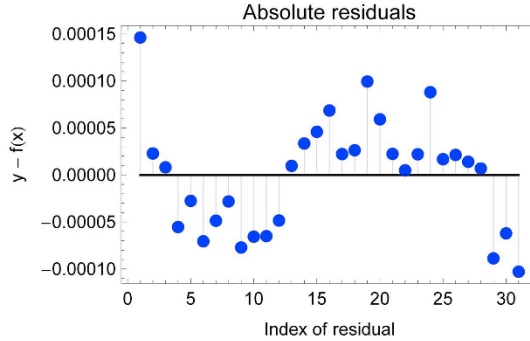
$$\eta = f(T) = a_1 \exp \left\{ \frac{a_2}{a_3 - T} \right\}$$

```
modelFunction=a1*Exp[a2/(a3-T)];
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.1},{a2,-500.0},{a3,150.0}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot"},xyErrorData,
  curveFitInfo,CurveFitOptionLabels -> labels];
```



From visual inspection the fit looks perfect and the residuals plot

```
CIP`CurveFit`ShowFitResult[{"AbsoluteResidualsPlot",
  "CorrelationCoefficient"},xyErrorData,curveFitInfo];
```

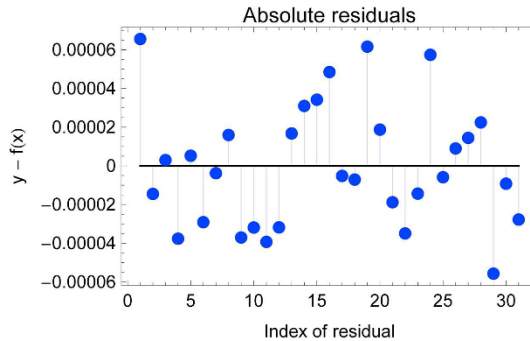



Out 1 : Correlation coefficient = 1.

reveals residuals that satisfactorily correspond to the experimental error of 0.0001 cP (this model function was historically found by Vogel after laborious linearization work, see [Vogel 1921]). Since a systematic pattern of deviations is still obvious two nearby improvements are finally tested which do not increase the number of parameters. First the initial factor is divided by T to try a combination with the inversely proportional approach tested earlier

$$\eta = f(T) = \frac{a_1}{T} \exp \left\{ \frac{a_2}{a_3 - T} \right\}$$

```
modelFunction=a1/T*Exp[a2/(a3-T)];
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.1},{a2,-500.0},{a3,150.0}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters];
CIP`CurveFit`ShowFitResult[{"AbsoluteResidualsPlot",
"CorrelationCoefficient"},xyErrorData,curveFitInfo];
```



Out 1 : Correlation coefficient = 1.

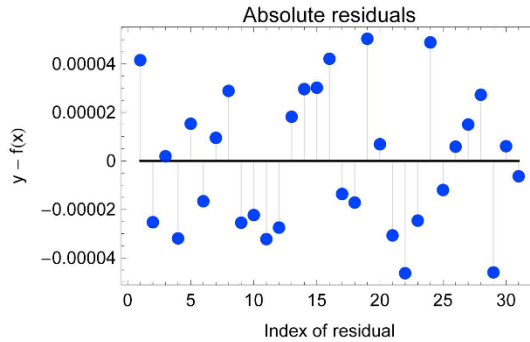
```
ShowFitResult[{"SDFit"}, xyErrorData, curveFitInfo];
```

Standard deviation of fit = 3.321×10^{-5}

and in addition the shift along the T axis is generalized:

$$\eta = f(T) = \frac{a_1}{a_3 - T} \exp \left\{ \frac{a_2}{a_3 - T} \right\}$$

```
modelFunction=a1/(a3-T)*Exp[a2/(a3-T)];
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.1},{a2,-500.0},{a3,150.0}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters];
CIP`CurveFit`ShowFitResult[{"AbsoluteResidualsPlot",
"CorrelationCoefficient"},xyErrorData,curveFitInfo];
```



Out 1 : Correlation coefficient = 1.

```
CIP`CurveFit`ShowFitResult[{"SDFit"}, xyErrorData, curveFitInfo];
```

Standard deviation of fit = 2.937×10^{-5}

The latter model function produces an absolutely convincing result: Systematic deviation patterns are vanished and the residuals are even below the estimated experimental error. This is also revealed by the χ^2_{red} value of

```
CIP`CurveFit`ShowFitResult[{"ReducedChiSquare"}, xyErrorData,
curveFitInfo];
```

Reduced chi-square of fit = 8.625×10^{-2}

which is considerably below 1 (this finding will be discussed in a subsequent chapter in combination with parameter errors). The optimum description of the data achieved by empirical construction may now be stated:

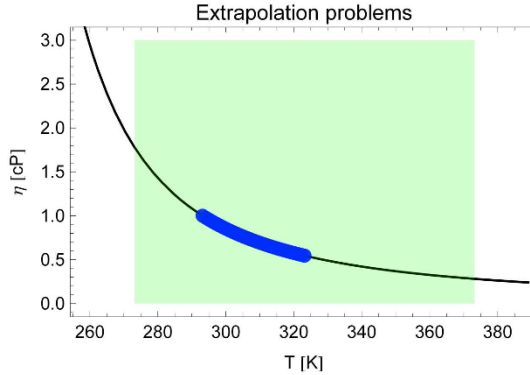
```
CIP `CurveFit `ShowFitResult[{"ModelFunction"},xyErrorData,
  curveFitInfo];
```

Fitted model function:

$$-\frac{19.3098e^{-\frac{200.831}{179.802-T}}}{179.802-T}$$

For a satisfactory description of the data a three-parameter model function is necessary compared to the two-parameter results (this view is also supported by the historical trend from Andrade to Vogel, see above). The final model function may be used for interpolation and extrapolation purposes. If the correlation coefficient is again inspected for the sketched trial and error model generation procedure its relative value corresponds to the true goodness of fit of each model (the better the model the closer is the correlation coefficient to one). But note that its absolute value is always very near to 1 so care has to be taken if a guessed model function is only cited with its correlation coefficient without any further information (which quite often occurs in practice) since this does not necessarily mean a good fit. For the water-viscosity data it is finally possible to precisely show what is meant by reasonable extrapolation with the following plot:

```
pureFunction=Function[x,CIP `CurveFit `CalculateFunctionValue[x,
  curveFitInfo]];
argumentRange={263.0,383.0};
plotRange={0.0,3.0};
plotStyle={{Thickness[0.005],Black}};
labels={"T [K]", "\[Eta] [cP]", "Extrapolation problems"};
extrapolationGraphics=
  CIP `Graphics `PlotXyErrorDataAboveFunctions[xyErrorData,
    {pureFunction},argumentRange,plotRange,plotStyle,labels];
intervalGraphics=Graphics[{RGBColor[0,1,0,0.2],
  Rectangle[{273.15,0.0},{373.15,3.0}]}];
Show[extrapolationGraphics,intervalGraphics]
```



The model function does not know that liquid water undergoes phase transitions if the temperature is lowered or raised beyond the illustrated background region in the diagram: Below 273.15 K water is solid matter (ice) with a practically infinite viscosity, above 373.15 K water is gaseous (vapor) with a dramatically reduced viscosity. To calculate a viscosity at 260 K is possible

```
argument=260;
CIP `CurveFit `CalculateFunctionValue[argument,curveFitInfo]
```

2.94556

but this value is not of this world. Whereas extrapolations around the data argument range may be helpful and sufficiently precise any large-scale extrapolations should always be regarded with suspicion. In summary it should be noted that the outlined construction strategy is very common for an educated guess of a model function. A combination of experience with mere trial and error is very often successful in two-dimensional curve fitting.

2.4 Problems and pitfalls

Linear as well as non-linear curve fitting was shown to be an optimization task (again note that the terms linear and non-linear denote the linearity or non-linearity of the model function with regard to its parameters a_1 to a_L , not the linear or non-linear dependence of the function value y on the argument value x): The global minimum of the $\chi^2(a_1, \dots, a_L)$ surface is to be found. As discussed in chapter 1 minimization procedures may fail. Failure leads to wrong estimates for the parameters' values and the parameters' errors or even a crash (i.e. an internal termination) of the whole fitting procedure.

Linear curve fitting implies the minimization of a parabolic $\chi^2(a_1, \dots, a_L)$ hyper surface that contains only one global minimum which can be calculated directly

by analytical means (see [Hamilton 1964], [Bevington 2002] or [Brandt 2002] for details). But this calculation involves a matrix inversion which can be a numerically ill-conditioned operation, i.e. problems may occur because computers can only calculate with a finite number of digits. These numerical problems can be tackled with state-of-the-art algorithms so failure usually happens in consequence of the implementation of deficient algorithms with missing safeguards against numerical instabilities. Since CIP is based on Mathematica which provides state-of-the-art algorithms linear curve fitting almost always works without problems. But there should be some awareness if alternative software applications are used as black boxes for linear curve fitting to avoid unnoticed pitfalls: There is a lot of dangerous stuff around - may it be commercial or free.

The situation with non-linear curve fitting is fundamentally different: Since $\chi^2(a_1, \dots, a_L)$ may be an arbitrarily difficult and complex curved hyper surface for a non-linear model function it may possess a plethora of minima. There is no way to directly calculate the global minimum by analytical means in principle. The $\chi^2(a_1, \dots, a_L)$ hyper surface can only be searched by iterative local minimization procedures that start at user-defined parameters' values and explore their surroundings (compare chapter 1 and [FitModelFunction] in the references). In addition to these principal issues the numerical problems sketched for linear curve fitting may be encountered as well or even in a more serious manner. So in practice there may be an evil mixture of problems - some that can be avoided by state-of-the-art software and others that can only be attributed to the nature of the fitting problem and may be tackled by specific strategies. Some practical problems together with possible solution strategies are outlined in the following.

2.4.1 Parameters' start values

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

The necessity of adequate parameters' start values may be illustrated by an example. Fifty xy-error data triples

```
numberOfData=50;
```

are simulated around the non-linear Gaussian-peak shaped function

$$y = f(x) = \frac{1}{2}x + 3 \exp\{-(x-4)^2\}$$

```
pureOriginalFunction=Function[x,0.5*x+3.0*Exp[-(x-4.0)^2]];
```

in the argument range [1.0, 7.0]

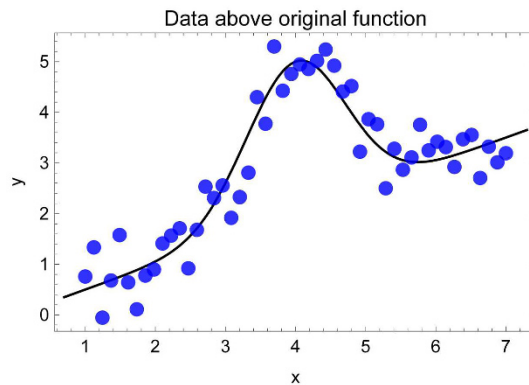
```
argumentRange={1.0,7.0};
```

with an absolute standard deviation of 0.5

```
standardDeviationRange={0.5,0.5};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
```

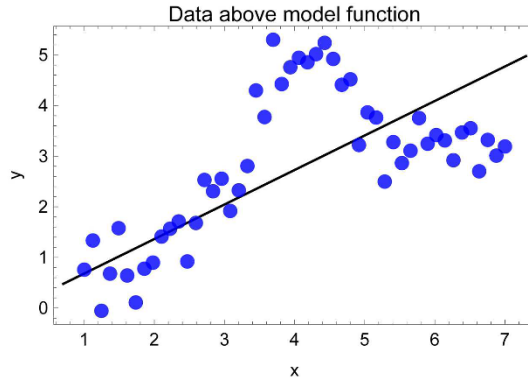
and finally plotted for visual inspection:

```
labels={"x","y","Data above original function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]
```



If the data are fitted with the corresponding model function with three parameters and the CIP default settings

```
modelFunction=a1*x+a2*Exp[-(x-a3)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];
```

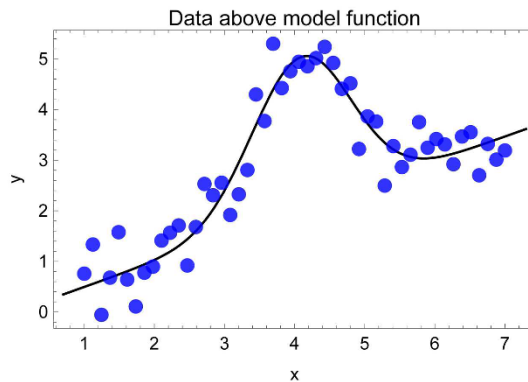


Fitted model function:

$$5.53945e^{-(25.4806+x)^2} + 0.681816x$$

the achieved result is simply wrong. What happened? Internally the FitModelFunction method generates random parameters' start values for the local minimization procedure - and these start values are simply inadequate in this case (but they may work perfectly in other fitting procedures). So start values for the parameters must be provided by hand. Since the true parameter values are 0.5, 3.0 and 4.0 (see above) everything works fine if parameters' start values are specified near the solution:

```
startParameters={{a1,0.4},{a2,3.1},{a3,3.9}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];
```



Fitted model function:

$$3.01029e^{-(4.09318+x)^2} + 0.495279x$$

A perfect fit is the result. It is well-known to practitioners that fitting Gaussian-peak shaped model functions requires a good guess for the parameter value in the exponential term: This start value may be deduced from the mere data in this case: The maximum is around $x = 4$ so use a value around 4 as a start value for parameter a_3 .

The worst case occurs if a_3 is chosen to be very unfavorable: Then the whole fitting procedure may crash (i.e. may internally be terminated) as shown in the following example:

```
startParameters={ {a1,0.4},{a2,3.1},{a3,-3.0}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters];
```

Underflow occurred in computation.

A value was calculated during the minimization process that was smaller than the smallest allowed value of the Mathematica system and therefore an underflow error message (and subsequent error messages) were generated. Note that this behavior can not simply be traced to a bad algorithm: The default Levenberg-Marquardt algorithm used by `FitModelFunction` for two-dimensional non-linear curve fitting is a state-of-the-art algorithm for this purpose. But it may fail in principle: It can not safeguard every possible calculation. It might be a good idea to simply change the minimization algorithm: An alternative minimization algorithm will usually generate a different outcome. That is why a library of algorithms is most often a severe advantage. But not in this case: If the algorithm is changed from Levenberg-Marquardt to Conjugate-Gradient

```
method={"ConjugateGradient"};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters,
  CurveFitOptionMethod -> method];
```

The line search decreased the step size to within tolerance specified by `AccuracyGoal` and `PrecisionGoal` but was unable to find a sufficient decrease in the norm of the residual.

a similar problem as before occurs in the line search subroutine of this algorithm. Another switch to the mere Gradient algorithm

```
method={"Gradient"};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters,
  CurveFitOptionMethod -> method];
```



```
CIP `CurveFit `ShowFitResult[{"ModelFunction"}, xyErrorData,
curveFitInfo];
```

Fitted model function:

$$13.0817e^{-(1.36437+x)^2} + 0.681692x$$

does not help either: The minimization procedure seems to have converged (since there are no error messages) but it stopped somewhere over the rainbow: The result is simply wrong. So the only practical solution is to provide good parameters' start values by hand, i.e. by ...

- ... **knowledge:** Parameters may be known to lie within defined intervals by experience or they may have a scientific meaning (i.e. they are theoretically well-defined) so that their values are approximately known in advance. In some cases a start value for a parameter may be deduced by visual inspection as in the example above for parameter a_3 in the exponential term.
- ... **trial and error:** Not a promising strategy but often the only practical possibility: It may be very exhaustive and disappointing but science is often more devoted to mere trial and error than scientists like to tell.

In the next section the trial and error case is tackled with more strategic approaches but these also can not solve the problem in principle.

2.4.2 How to search for parameters' start values

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

To get good parameters' start values a global search of the parameter space is necessary: A huge task! In chapter 1 different strategies for a global search were discussed like a grid or a random search. CIP implements a purely random search strategy as an option for the GetStartParameters method of the CIP CurveFit package with search type "Random":

```
searchType="Random";
```

For the curve fitting task outlined of the last subsection

```
numberOfData=50;
pureOriginalFunction=Function[x, 0.5*x+3.0*Exp[-(x-4.0)^2]];
argumentRange={1.0, 7.0};
standardDeviationRange={0.5, 0.5};
```

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
modelFunction=a1*x+a2*Exp[-(x-a3)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
```

a parameters' search space is defined by the individual intervals of each parameter

```
parameterIntervals={{0.0,10.0},{0.0,10.0},{0.0,10.0}};
```

with a 100 random trial points:

```
numberOfTrialPoints=100;
```

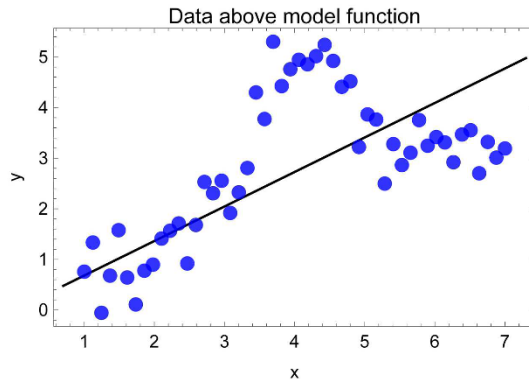
If the GetStartParameters method is called with these settings

```
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,CurveFitOptionSearchType -> searchType,
CurveFitOptionNumberOfTrialPoints -> numberOfTrialPoints]
```

```
{{a1,0.670859},{a2,7.48994},{a3,8.29601}}
```

the resulting parameters' start values correspond to the smallest value of $\chi^2(a_1, a_2, a_3)$ that was detected by random. These start values are now used as an input for the model function fit by setting the CurveFitOptionStartParameters option with the result:

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];
```



Fitted model function:

$$97.9599e^{-(17.502+x)^2} + 0.681816x$$

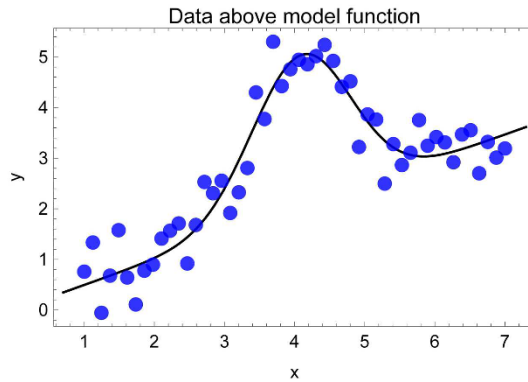
The result is still not correct: 100 trial points do not lead to sufficiently precise parameters' start values since the random grid is too coarsely meshed. Therefore their number is increased tenfold to 1000 and the search is repeated:

```
numberOfTrialPoints=1000;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,CurveFitOptionSearchType -> searchType,
CurveFitOptionNumberOfTrialPoints -> numberOfTrialPoints]
```

```
{{a1,0.679172},{a2,1.7166},{a3,4.48335}}
```

With the new start values

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];
```



Fitted model function:

$$3.01029e^{-(4.09318+x)^2} + 0.495279x$$

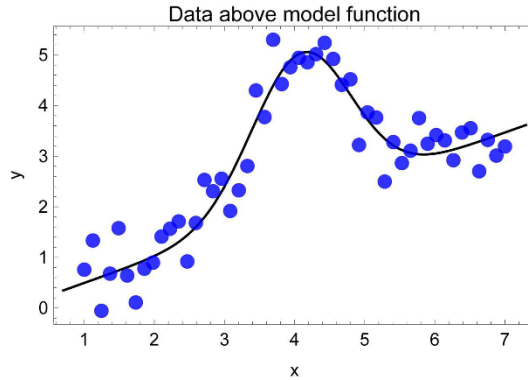
a successful fit is finally obtained: The determined start values were precise enough for the local minimization algorithm to converge to the global minimum of $\chi^2(a_1, a_2, a_3)$. Although this may seem promising it again should be noticed that a random search is a rather limited option in general: Since the parameter space becomes really large with an increasing number of parameters a random search within tolerable periods of time will be likely to fail. The glimmer of hope of chapter 1 in this desperate situation were evolutionary algorithms. Method `GetStartParameters` uses the differential-evolution algorithm via Mathematica's `NMinimize` command as its default global search strategy (see `[NMinimize/NMaximize]` in the references) which also proves to be successful for the current task:

```
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  parameterIntervals]
```

```
{{a1,0.460666},{a2,3.67567},{a3,4.12436}}
```

A fit with the obtained start parameters

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
  xyErrorData,curveFitInfo];
```



Fitted model function:

$$3.01029e^{-(4.09318+x)^2} + 0.495279x$$

shows that the evolutionary search was able to determine start values in the proximity of the global minimum of $\chi^2(a_1, a_2, a_3)$ which were close enough for a successful local refinement.

2.4.3 More difficult curve fitting problems

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

Extracting the correct model function from experimental data may be arbitrarily difficult up to impossible due to the nature of the curve fitting problem. To demonstrate an example fifty xy-error data triples

```
numberOfData=50;
```

with a very high precision (absolute standard deviation of 0.001)

```
standardDeviationRange={0.001,0.001};
```

are generated in an argument range [1, 8]

```
argumentRange={1.0,8.0};
```

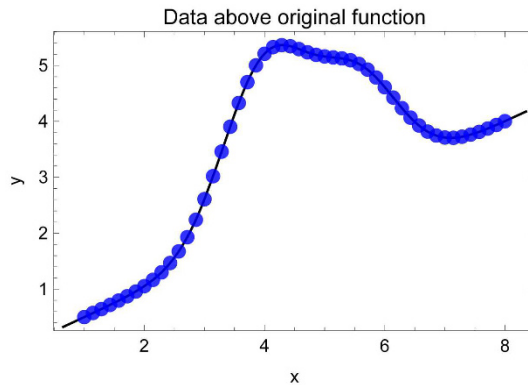
around a model function with two Gaussian peaks in close proximity (around $x = 4$ and $x = 5.5$)

$$y = f(x) = \frac{1}{2}x + 3\exp\{-(x-4)^2\} + 2\exp\{-(x-5.5)^2\}$$

```
pureOriginalFunction=
Function[x, 0.5*x+3.0*Exp[-(x-4.0)^2]+2.0*Exp[-(x-5.5)^2]];
```

where the smaller one around $x = 5.5$ appears to be the shoulder of the other:

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above original function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]
```



With the previous discussions in mind it should be obvious that a successful curve fitting procedure needs very good start values for the parameters in this case. Again at least start values for the parameters in the exponentials could be obtained by mere visual inspection of the generated data (peaks around $x = 4$ and $x = 5.5$) but a more general strategy is explored that uses the advised start-parameter search on the basis of an evolutionary algorithm with the `GetStartParameters` method of the CIP CurveFit package. With the 5-parameter model function

```
modelFunction=a1*x+a2*Exp[-(x-a3)^2]+a4*Exp[-(x-a5)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3,a4,a5};
```

and a well defined parameters' search space

```
parameterIntervals=
  {{0.0,10.0},{0.0,10.0},{0.0,10.0},{0.0,10.0},{0.0,10.0}};
```

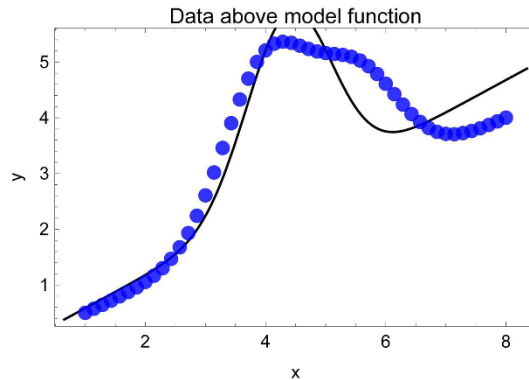
the proposed parameters' start values for the fit procedure are:

```
maximumNumberOfIterations=10;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,
CurveFitOptionMaximumIterations -> maximumNumberOfIterations]
```

```
{{a1,0.443407},{a2,4.81041},{a3,4.83269},{a4,0.0454534},{a5,4.45465}}
```

A fit with these parameters' start values

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];
```



Fitted model function:

$$-0.173247e^{-(42.6798+x)^2} + 3.33414e^{-(4.38782+x)^2} + 0.584755x$$

leads to an unsatisfying result. Obviously the parameters' start values search was not successful - remember that there is no guarantee for an evolutionary strategy to succeed. This failure might be attributed to the applied setting of the internal number of iterations (i.e. the number of generations for evolution) to only 10. In this particular case the parameter space should be explored more thoroughly with an increased number of iterations (note that this number must always be restricted to balance between accuracy and speed):

```

maximumNumberOfIterations=100;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,
CurveFitOptionMaximumIterations -> maximumNumberOfIterations]

```

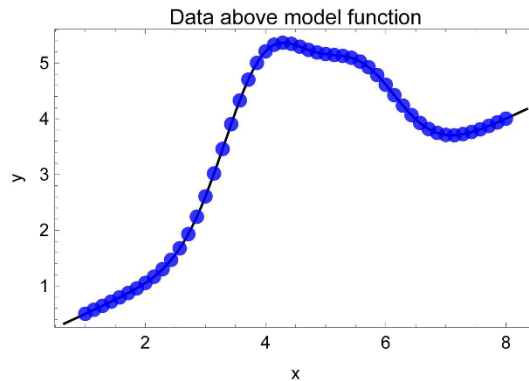
```
{a1,0.535511},{a2,1.39872},{a3,5.55438},{a4,2.80221},{a5,4.13868}}
```

With the improved parameters' start values the curve fitting procedure

```

curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction"},
xyErrorData,curveFitInfo];

```



Fitted model function:

$$2.00039e^{-(5.49993+x)^2} + 2.99947e^{-(4.00006+x)^2} + 0.5x$$

is successful: A perfect fit is obtained. The sketched curve fitting problem will certainly become more difficult if the two Gaussian peaks are moved together, e.g.

$$y = f(x) = \frac{1}{2}x + 3 \exp\{-(x-4)^2\} + 2 \exp\{-(x-4.5)^2\}$$

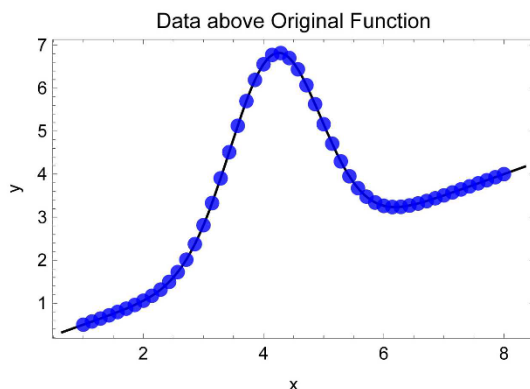
```

pureOriginalFunction=
Function[x, 0.5*x+3.0*Exp[-(x-4.0)^2]+2.0*Exp[-(x-4.5)^2]];

```

where the two peaks now are closely neighbored around $x = 4$ and $x = 4.5$. After xy-error data generation as before a visual inspection shows


```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above Original Function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]
```



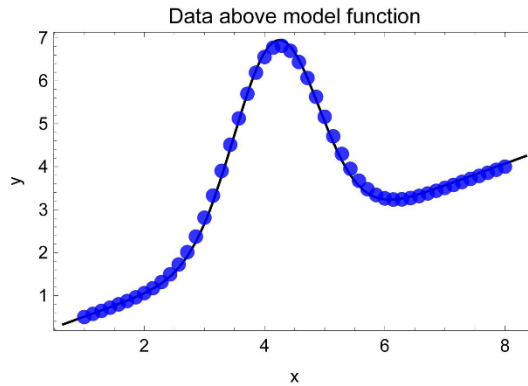
that the shoulder becomes invisible and only one merged peak appears. Note that without an a priori knowledge about the two existing peaks (the data are artificial) only one peak would be anticipated. If again the parameters' start value search is used with an insufficient number of iterations

```
maximumNumberOfIterations=20;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,
CurveFitOptionMaximumIterations -> maximumNumberOfIterations]
```

```
{{a1,0.554496},{a2,4.20733},{a3,4.10765},{a4,3.44001},{a5,9.99968}}
```

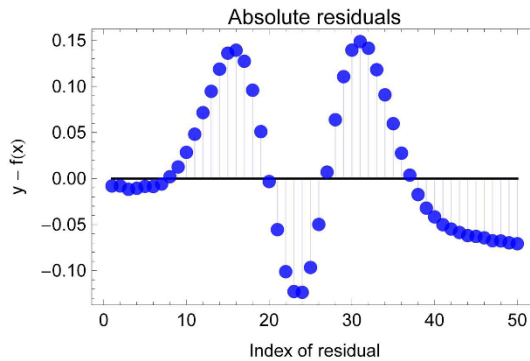
the results are dubious, i.e. values are too close to the search boundaries. A fit with these start values give evidence for this assessment:

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction",
"AbsoluteResidualsPlot"},xyErrorData,curveFitInfo];
```



Fitted model function:

$$158.045e^{-(58.0059+x)^2} + 4.8003e^{-(-4.19422+x)^2} + 0.508776x$$



The second Gaussian peak is sent to infinity. This leads to a systematic deviation pattern of the residuals which is a clear indication that something is missed. A further refinement of the parameters' space exploration becomes necessary with an additional increase of the number of iterations:

```
maximumNumberOfIterations=100;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,
CurveFitOptionMaximumIterations -> maximumNumberOfIterations]
```

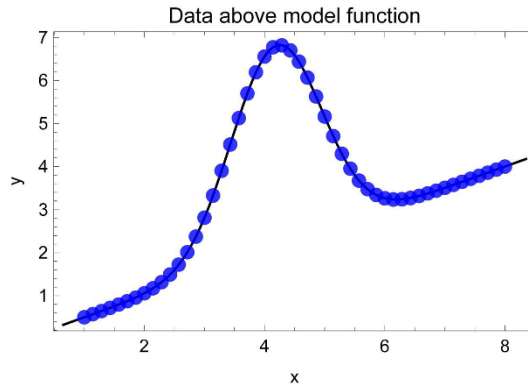
```
{{a1,0.505924},{a2,4.76675},{a3,4.18593},{a4,0.125418},{a5,5.46316}}
```

The following fit

```

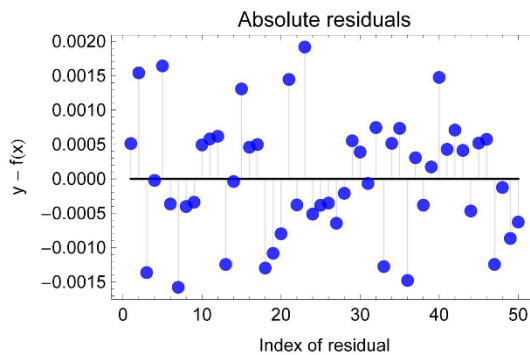
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","ModelFunction",
"AbsoluteResidualsPlot"},xyErrorData,curveFitInfo];

```



Fitted model function:

$$2.00853e^{-(4.49914+x)^2} + 2.99136e^{-(3.99927+x)^2} + 0.500001x$$

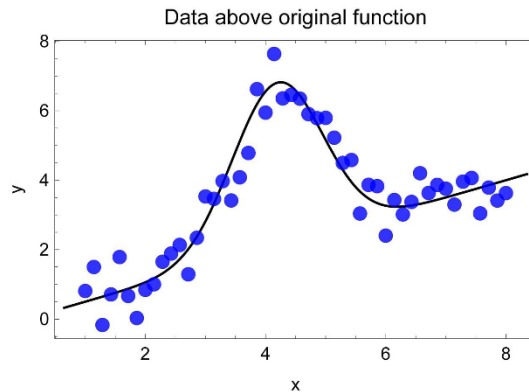


now leads to a satisfactory result. This successful outcome was invoked by a continuously intensified brute-force strategy that led to an enhanced thoroughness of parameters' space exploration. In summary it is a remarkable fact that data analysis is able to reveal invisible peaks that would not be assumed by mere visual inspection but only if they are known to be there. On the other hand subtle interpretation problems will emerge if things become slightly more difficult in the case of less precise data. For an illustration low-precision data are generated with a high absolute standard deviation of 0.6 around the last example function:

```

standardDeviationRange={0.6,0.6};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above original function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]

```



The thorough parameters' start value search with again a large number of evolutionary steps

```

maximumNumberOfIterations=100;
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals,
CurveFitOptionMaximumIterations -> maximumNumberOfIterations]

```

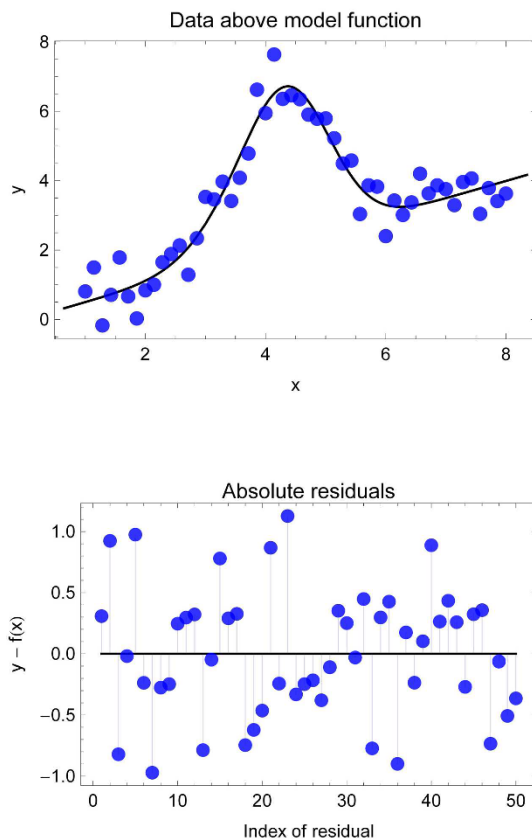
```
{{a1,0.512016},{a2,4.55652},{a3,4.28594},{a4,0.302777},{a5,3.01461}}
```

and a following fit

```

curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"SDFit","ModelFunction"},xyErrorData,curveFitInfo];

```



Standard deviation of fit = 5.478×10^{-1}

Fitted model function:

$$4.25758e^{-(4.37855+x)^2} + 0.725734e^{-(3.3966+x)^2} + 0.49908x$$

lead to a result of good quality with effectively two different peaks. But the precision of peak detection is no longer satisfying. Moreover this result is no longer convincing if alternatives are taken into consideration. This can be shown with an alternative fit of the corresponding model function with one Gaussian peak which would be assumed by mere visual inspection:

```
modelFunction=a1*x+a2*Exp[-(x-a3)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
```

With adequate start values

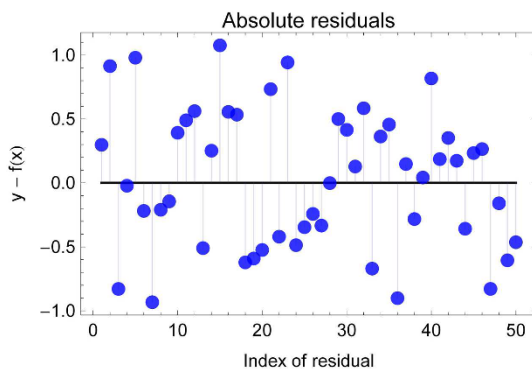
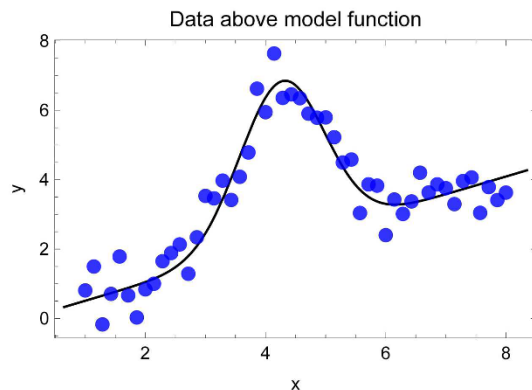
```
parameterIntervals={{0.0,10.0},{0.0,10.0},{0.0,10.0}};
```

```
startParameters=CIP`CurveFit`GetStartParameters[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
parameterIntervals]
```

```
{{a1,0.523746},{a2,5.85935},{a3,4.12508}}}
```

the alternative one-peak fit

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"SDFit","ModelFunction"},xyErrorData,curveFitInfo];
```



Standard deviation of fit = 5.527×10^{-1}

Fitted model function:

$$4.65303e^{-(-4.27207 + x)^2} + 0.511526x$$

leads to a result of comparable quality (the standard deviations of the fits are nearly identical and the residuals patterns are equally good): The latter model function should be preferred according to Occam's razor since it contains less parameters (unless the existence of two peaks is certainly known in advance). Depending on the precision of the data and the nature of the fitting problem severe ambiguities can appear in data analysis. In the last case peaks may be found or may be argued for that can not be supported by the mere data in the light of alternative models. So all data analysis procedures are prone to be misused for the sake of a scientist's mere opinion and not the truth (where the scientist is always assumed to pursue the most noble intentions). As a rule of thumb an adequate distrust is indicated for statements like *it is clearly shown by thorough data analysis that ..* Curve fitting should always be data driven and it should not be tried to get more out of them than possible. The old and latent tendency to overstretch data analysis once led to the famous sentence by John von Neumann: *With four parameters I can fit an elephant, and with five I can make him wiggle his trunk* ([Dyson 2004]).

2.4.4 Inappropriate model functions

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
```

Model functions may be unfavorable up to simply wrong. An example of the latter is demonstrated as follows: Fifty fairly precise data

```
numberOfData=50;
```

with an absolute standard deviation of 0.01

```
standardDeviationRange={0.01,0.01};
```

are generated in the argument range [1, 5]

```
argumentRange={1.0,5.0};
```

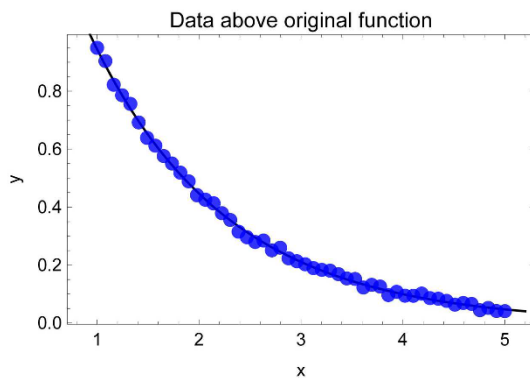
around function

$$y = f(x) = 2 \exp\{-1.5x\}$$

```
pureOriginalFunction=Function[x,2.0*Exp[-0.75*x]];
```

to give

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above original function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]
```



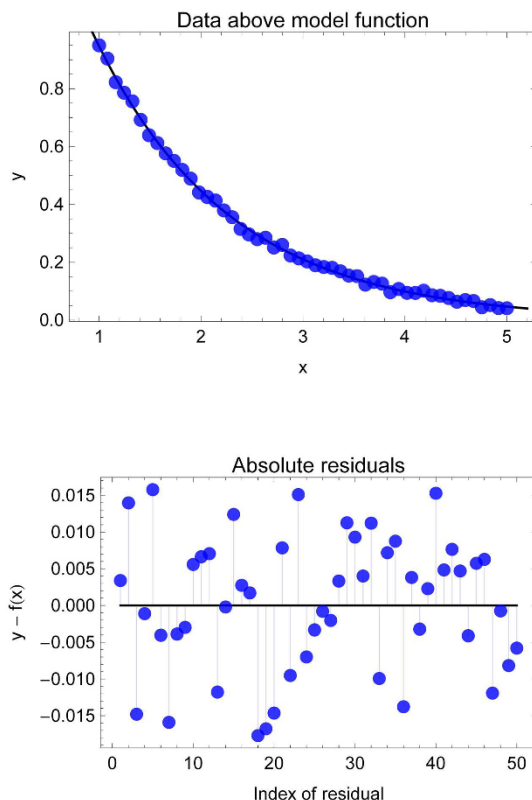
For the fit we use an empirical model function constructed without to much meditation:

$$y = f(x) = a_1 \exp \{-a_2 x + a_3\}$$

```
modelFunction=a1*Exp[-a2*x+a3];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
```

The result

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction];
ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ModelFunction"},xyErrorData,curveFitInfo];
```

Fitted model function:

$$1.91781e^{0.0463659 - 0.752542x}$$

appears to be a perfect fit. So what is wrong with the model function? The answer is that it contains redundant parameters since parameters a_1 and a_3 essentially mean the same: They are both mere prefactors to the exponential term

$$y = f(x) = a_1 \exp \{-a_2 x + a_3\} = a_1 \exp \{a_3\} \exp \{-a_2 x\}$$

and therefore they are arbitrary. Only their product is the true prefactor used to generate the data. All infinite other combinations of values resulting to the same prefactor would be valid as well. Although redundant parameters can always be avoided by proper inspection of the model function they do occur easily if non-mathematicians (i.e. the overwhelming majority of scientists) construct difficult empirical models. Usually the fitting algorithms simply crash if redundant parameters are defined in a model function. It is only due to Mathematica's algorithmic safeguards that lead to an arbitrary but correct result. A more subtle problem occurs if the model function is correct but simply inappropriate to the data since it tries to

extract information which is simply not there. This may be shown with fifty fairly precise data with an absolute standard deviation of 0.1

```
standardDeviationRange={0.1, 0.1};
```

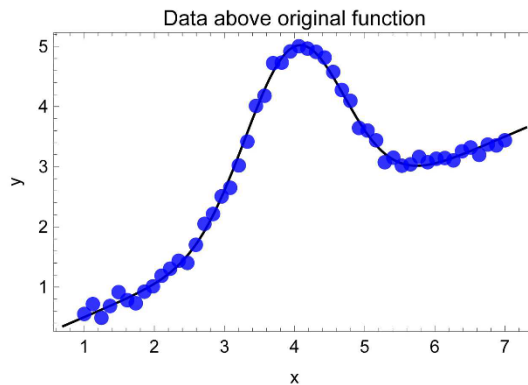
in the argument range [1, 7]

```
argumentRange={1.0, 7.0};
```

around one Gaussian peak:

$$y = f(x) = \frac{1}{2}x + 3 \exp\{-(x-4)^2\}$$

```
pureOriginalFunction=Function[x, 0.5*x+3.0*Exp[-(x-4.0)^2]];
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange, numberOfData, standardDeviationRange];
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction, labels]
```



A model function with two Gaussian peaks is prepared

```
modelFunction=a1*x+a2*Exp[-(x-a3)^2]+a4*Exp[-(x-a5)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1, a2, a3, a4, a5};
```

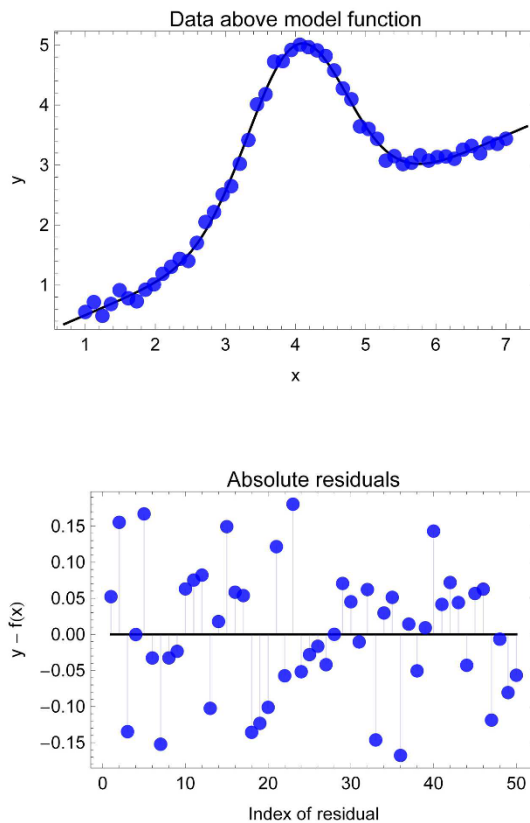
which inevitably tries to extract two Gaussian peaks from the data which just contain one peak. After an successful search for parameters' start values

```
parameterIntervals=
  {{0.0,10.0},{0.0,10.0},{0.0,10.0},{0.0,10.0},{0.0,10.0}};
startParameters=GetStartParameters[xyErrorData,modelFunction,
  argumentOfModelFunction,parametersOfModelFunction,
  parameterIntervals]
```

```
{{a1,0.533351},{a2,0.764096},{a3,4.36781},{a4,2.40032},{a5,3.9497}}
```

the following fit

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
  "ModelFunction"},xyErrorData,curveFitInfo];
```



Fitted model function:

$$2.50893e^{-(-4.01878+x)^2} + 0.489941e^{-(-4.01878+x)^2} + 0.499246x$$

shows what happened: The same peak was found twice with arbitrary prefactors that only have meaning as a sum. Again note: If alternative software to CIP/Mathematica is used the fitting algorithms usually crash if a model function is inappropriate as outlined in the latter example.

2.5 Parameters' errors

```
Clear["Global`*"];
<<CIP`ExperimentalData`
<<CIP`CurveFit`
```

The second most important information that may be extracted from a successful curve fitting procedure in accordance with the optimum estimates of the parameters' values are estimates of the parameters' errors.

2.5.1 Correction of parameters' errors

Since the xy-error data are biased by errors these errors propagate to the errors of the estimated parameters' values: The parameters' errors therefore are deduced from the data's errors. This is certainly the best procedure if the data's errors are true experimentally obtained errors, e.g. each y value is measured multiple times and then reported as the statistical mean y_i with the statistical standard deviation of the mean σ_i for an argument value x_i . But often the reported errors σ_i can only be regarded as rough estimates of the true errors. Moreover these estimates are usually overestimated since scientists tend to be cautious: A bigger error is the better error if the error is not known precisely. Then of course the resulting parameters' errors of a model function fit are also overestimated. As an example the water-viscosity data are inspected again (compare above):

```
xyErrorData=CIP`ExperimentalData`GetWaterViscosityXyErrorData[];
modelFunction=a1/(a3-T)*Exp[a2/(a3-T)];
argumentOfModelFunction=T;
parametersOfModelFunction={a1,a2,a3};
startParameters={a1,0.1},{a2,-500.0},{a3,150.0}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters];
CIP`CurveFit`ShowFitResult[{"ReducedChiSquare","ParameterErrors"},
xyErrorData,curveFitInfo];
```

Reduced chi-square of fit = 8.625×10^{-2}

	Value	Standard error	Confidence region
Parameter a1 =	-19.3098	0.108964	{-19.4208, -19.1989}
Parameter a2 =	-200.831	1.86845	{-202.734, -198.929}
Parameter a3 =	179.802	0.445257	{179.348, 180.255}

The χ^2_{red} value of 0.086 indicates that the fitted residuals are fair below the corresponding errors σ_i of the y_i values since χ^2_{red} should be close to 1 for a good fit with good data's errors. Consequently the data's errors should be decreased for a resulting χ^2_{red} value near 1. A correction for the data's errors may be calculated when they are assumed to be only weights of the y_i values and not their true statistical errors (see above). The FitModelFunction method can be told to estimate parameters' errors with the corrected and not the original errors by changing the option CurveFitOptionVarianceEstimator from its default value to "ReducedChiSquare":

```
varianceEstimator="ReducedChiSquare";
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionVarianceEstimator -> varianceEstimator];
CIP`CurveFit`ShowFitResult[{ "ReducedChiSquare", "ParameterErrors" },
xyErrorData, curveFitInfo];
```

Reduced chi-square of fit = 8.625×10^{-2}

	Value	Standard error	Confidence region
Parameter a1 =	-19.3098	0.0320015	{-19.3424, -19.2772}
Parameter a2 =	-200.831	0.548743	{-201.39, -200.272}
Parameter a3 =	179.802	0.130767	{179.669, 179.935}

The parameters' standard errors and their confidence regions are reduced by more than a factor of 3 in comparison to the result before. The outlined error correction is often used as a standard procedure for curve fitting. But in practice it simply depends on the problem and the scientist's mood to use the cautious (higher) error estimates for all subsequent derivations as well.

2.5.2 Confidence levels of parameters' errors

```
Clear["Global`*"];
<<CIP`ExperimentalData`
<<CIP`CurveFit`
```

Another important option that may be modified for the estimation of parameters' errors is their level of confidence which affects the width of their confidence regions. With the default setting of 68.3% the parameters' confidence regions correspond to the standard errors, i.e. a confidence region spans the interval $[a_i - \sigma_{a_i}, a_i + \sigma_{a_i}]$ where σ_{a_i} is the standard error of parameter a_i . In many cases a higher confidence

level of e.g. 95% or 99% is required. This may be specified with option `CurveFitOptionConfidenceLevel` of method `FitModelFunction`. Here a confidence level of 99.9%

```
confidenceLevelOfParameterErrors=0.999;
```

is used for the water-viscosity fit for with the corrected errors (see previous section):

```
xyErrorData=CIP `ExperimentalData `GetWaterViscosityXyErrorData[];
modelFunction=a1/(a3-T)*Exp[a2/(a3-T)];
argumentOfModelFunction=T;
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.1},{a2,-500.0},{a3,150.0}};
varianceEstimator="ReducedChiSquare";
curveFitInfo=CIP `CurveFit `FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionVarianceEstimator -> varianceEstimator,
CurveFitOptionConfidenceLevel -> confidenceLevelOfParameterErrors];
CIP `CurveFit `ShowFitResult[{"ReducedChiSquare","ParameterErrors"},
xyErrorData,curveFitInfo];
```

Reduced chi-square of fit = 8.625×10^{-2}

	Value	Standard error	Confidence region
Parameter a1 =	-19.3098	0.0320015	{-19.4274, -19.1922}
Parameter a2 =	-200.831	0.548743	{-202.847, -198.815}
Parameter a3 =	179.802	0.130767	{179.321, 180.282}

Note that the standard errors are not affected since they are related to the standard confidence level of 68.3% but the confidence regions increased considerably: Now it can be assured with a probability of 99.9% that the parameters' values are within the denoted regions.

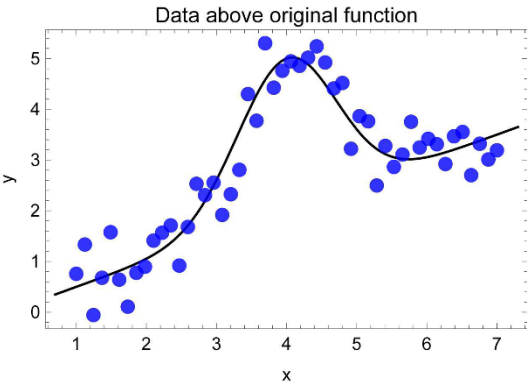
2.5.3 Estimating the necessary number of data

```
Clear["Global`*"];
<<CIP `CalculatedData `
<<CIP `Graphics `
<<CIP `CurveFit `
```

An practically important issue related to the parameters' errors is the following: A theoretical model function with well-defined parameters is known. A specific measurement process with its intrinsic measurement errors is available. How many experimental data in a defined argument range must be measured to get a reasonable statement about a parameters' value with a specific level of confidence? To get an

impression the Gaussian-peak shaped model function is taken again as an example. If a measurement process imposes an absolute error of 0.5 on each measurement the following parameters' errors and confidence regions are obtained for fifty (x_i, y_i, σ_i) data triples in the argument range [1.0, 7.0] with a confidence level of 68.3%:

```
numberOfData=50;
standardDeviationRange={0.5,0.5};
pureOriginalFunction=Function[x,0.5*x+3.0*Exp[-(x-4.0)^2]];
argumentRange={1.0,7.0};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above original function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels]
```



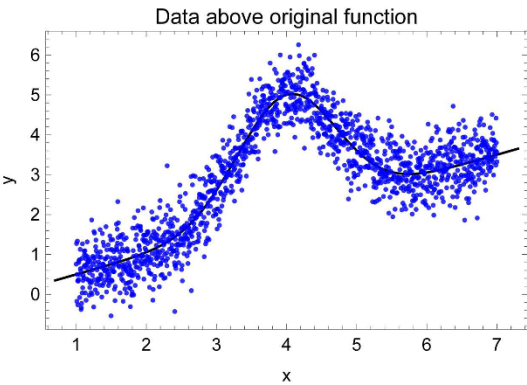
```
modelFunction=a1*x+a2*Exp[-(x-a3)^2];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.4},{a2,2.9},{a3,4.1}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters];
CIP`CurveFit`ShowFitResult[{"ModelFunction","ReducedChiSquare",
"ParameterErrors"},xyErrorData,curveFitInfo];
```

Fitted model function:
 $3.01029e^{-(4.09318+x)^2} + 0.495279x$
Reduced chi-square of fit = 8.17×10^{-1}

	Value	Standard error	Confidence region
Parameter a1	= 0.495279	0.0205374	{0.474521, 0.516038}
Parameter a2	= 3.01029	0.196361	{2.81182, 3.20877}
Parameter a3	= 4.09318	0.0528048	{4.0398, 4.14655}

If the number of data is increased the parameters' values will become more precise and the parameters' errors and their related confidence regions are reduced:

```
numberOfData=1500;
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
pointSize=0.01;
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,
pureOriginalFunction,labels,GraphicsOptionPointSize -> pointSize]
```



```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters];
CIP`CurveFit`ShowFitResult[{"ModelFunction","ReducedChiSquare",
"ParameterErrors"},xyErrorData,curveFitInfo];
```

Fitted model function:

$$2.97527e^{-(-3.99917+x)^2} + 0.502659x$$

Reduced chi-square of fit = 9.819×10^{-1}

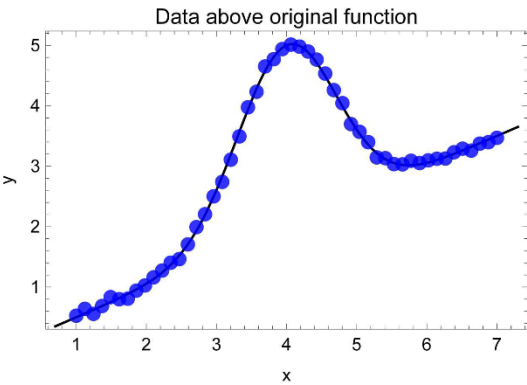
	Value	Standard error	Confidence region
Parameter a1	= 0.502659	0.00374083	{0.498917, 0.506401}
Parameter a2	= 2.97527	0.0352989	{2.93996, 3.01058}
Parameter a3	= 3.99917	0.00966191	{3.9895, 4.00883}

As a second alternative another measurement process may be available with a decreased intrinsic error that it imposes on the data (here the absolute error is reduced by a factor of 10 from 0.5 to 0.05):

```
numberOfData=50;
standardDeviationRange={0.05,0.05};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
```



```
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData,  
pureOriginalFunction,labels]
```



```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,  
modelFunction,argumentOfModelFunction,parametersOfModelFunction,  
CurveFitOptionStartParameters -> startParameters];  
CIP`CurveFit`ShowFitResult[{"ModelFunction","ReducedChiSquare",  
"ParameterErrors"},xyErrorData,curveFitInfo];
```

Fitted model function:
 $2.99923e^{-(4.00939+x)^2} + 0.499635x$
Reduced chi-square of fit = 8.163×10^{-1}

	Value	Standard error	Confidence region
Parameter a1	= 0.499635	0.0020301	{0.497583, 0.501687}
Parameter a2	= 2.99923	0.01941	{2.97961, 3.01885}
Parameter a3	= 4.00939	0.00529798	{4.00403, 4.01474}

Improved estimates of the parameters' values as well as decreased parameters' errors and smaller confidence regions are the result. Unfortunately the latter possibility of an alternative measurement process with increased precision is only rarely encountered in practice. So the only method of choice is usually to increase the number of data which means more time and more money. To estimate this critical quantity in advance the simulation of the necessary number of experimental data is always helpful and indicated. The CIP CurveFit package provides the GetNumberOfData method to fulfill this task: This method tries to detect the necessary number of data necessary to achieve a desired width of the confidence region of a specified parameter for a specified confidence level by an iterative process. If a width of the confidence region of 0.01 for parameter a_3 is desired

```
desiredWidthOfConfidenceRegion=0.01;
indexOfParameter=3;
```

the necessary number of data for the latter example would be

```
numberOfData=CIP`CurveFit`GetNumberOfData[
desiredWidthOfConfidenceRegion,indexOfParameter,
pureOriginalFunction,argumentRange,standardDeviationRange,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters]
```

57

For a halved confidence region of 0.005

```
desiredWidthOfConfidenceRegion=0.005;
```

the number of data must be increased to about 221:

```
numberOfData=CIP`CurveFit`GetNumberOfData[
desiredWidthOfConfidenceRegion,indexOfParameter,
pureOriginalFunction,argumentRange,standardDeviationRange,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters]
```

221

Note that there is a strong non-linear relation between the necessary number of data and the width of a confidence region: To half the width of a confidence region in value there is a considerable increase of the number of data necessary in general.

2.5.4 Large parameters' errors and educated cheating

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`CurveFit`
```

For specific model functions very precise experimental data are necessary to estimate its parameters' values with a sufficient precision. A good example are power laws that play an important role in different areas of science like critical phenomena or the analysis of biological (scale-free) networks. A power law of the form

$$y = f(x) = a_1 |x - a_2|^{-a_3}$$

that diverges at $x = a_2$ with a so called critical exponent a_3 will be discussed in the following. Power law fits are often used to prove or reject a specific theoretical prediction whereupon the critical exponent a_3 enjoys the highest attention: Therefore this parameter is to be estimated with an utmost precision. For a power law fit a search for parameters' start values is not necessary in most cases since all parameters are approximately known in advance from theory or visual inspection of the data: The critical exponent a_3 comes from theory, the location of the divergence a_2 may be directly deduced from the data so only the prefactor a_1 is in question. As an example fifty high precision normally distributed data

```
numberOfData=50;
```

will be generated around the power law

$$y = f(x) = 2|x - 10|^{-0.63}$$

```
pureOriginalFunction=Function[x,2.0*Abs[x-10.0]^(-0.63)];
```

in the argument range [8.0, 9.9]

```
argumentRange={8.0,9.9};
```

with a relative standard deviation of 0.1%:

```
errorType="Relative";
standardDeviationRange={0.001,0.001};
```

The arguments will be spaced by a logarithmic scale to push more data into the divergence region (as is usually performed by a proper design of experiment):

```
argumentDistance="LogLargeToSmall";
```

The xy-error data are generated with method `GetXyErrorData` of the CIP `CalculatedData` package:

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange,
CalculatedDataOptionErrorType->errorType,
CalculatedDataOptionDistance->argumentDistance];
```

The model function to fit is set in accordance

```
modelFunction=a1*Abs[x-a2]^(-a3);
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
```

The necessary start parameters are chosen to be near the true parameters:

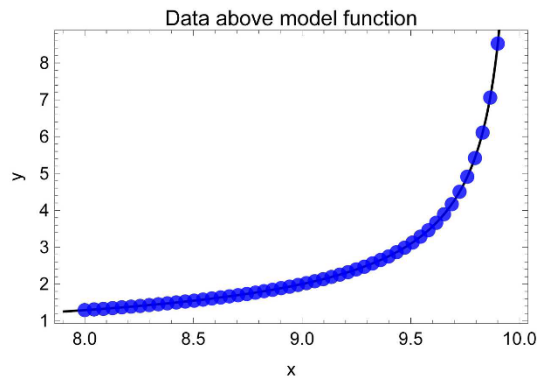
```
startParameters={{a1,1.9},{a2,9.99},{a3,-0.6}};
```

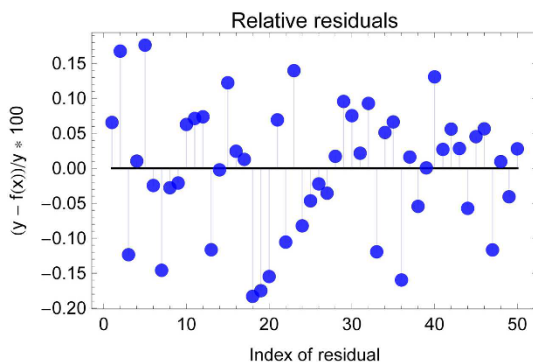
A high confidence level of 99.9% is advised for the confidence region of the parameters:

```
confidenceLevelOfParameterErrors=0.999;
```

For these simulated data a perfect fit results:

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters,
CurveFitOptionConfidenceLevel->
confidenceLevelOfParameterErrors];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
curveFitInfo];
```





Standard deviation of fit = 1.78×10^{-3}

Reduced chi-square of fit = 8.514×10^{-1}

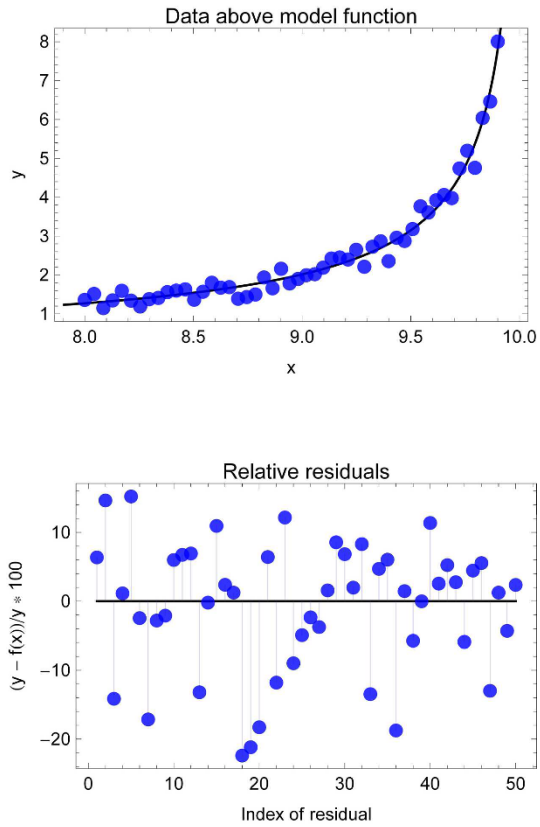
	Value	Standard error	Confidence region
Parameter a1 =	2.0006	0.000557139	{1.99864, 2.00255}
Parameter a2 =	10.0004	0.000295439	{9.99932, 10.0014}
Parameter a3 =	0.630468	0.000458764	{0.628857, 0.632078}

The critical exponent a_3 is found to be in a small confined interval $[0.629, 0.632]$ around 0.63 with a high probability of 99.9%. If a theoretical model would predict the value of 0.63 this fit would rightly be regarded as a *strong experimental evidence* (by cautious scientists) up to a *convincing experimental proof* (by more enthusiastic ones). Unfortunately experimental data for power law fits are often far less precise. This has a dramatic influence on the confidence region of the critical exponent a_3 as shown in the next example. The relative error of the data is increased by a factor of 100 to 10%:

```
standardDeviationRange={0.1,0.1};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange,
CalculatedDataOptionErrorType -> errorType,
CalculatedDataOptionDistance -> argumentDistance];
```

The corresponding fit

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionConfidenceLevel ->
confidenceLevelOfParameterErrors];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
curveFitInfo];
```



Standard deviation of fit = 1.779×10^{-1}
Reduced chi-square of fit = 8.512×10^{-1}

	Value	Standard error	Confidence region
Parameter a1 =	2.0764	0.0830715	{1.78482, 2.36797}
Parameter a2 =	10.0444	0.043694	{9.89101, 10.1977}
Parameter a3 =	0.68482	0.0580304	{0.481139, 0.888501}

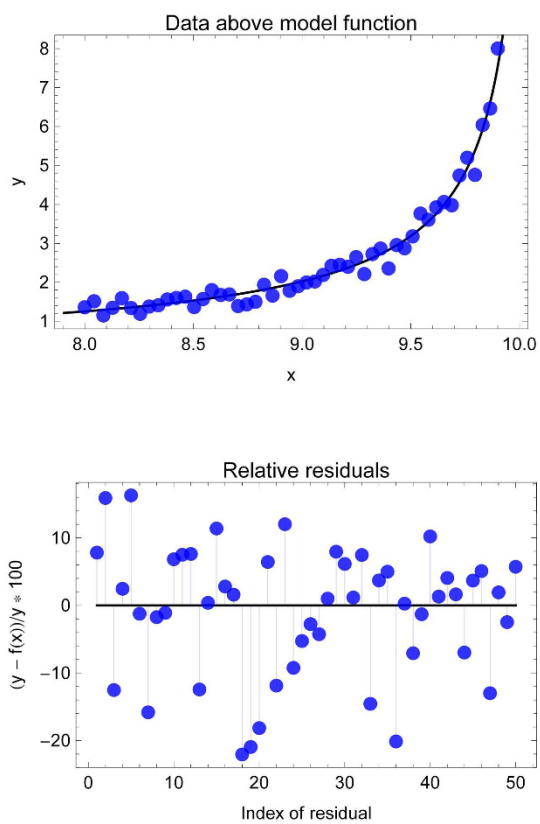
again looks perfect but the confidence region of the critical exponent a_3 is found to be nearly as large ($0.89 - 0.48 = 0.41$) as the absolute value of the parameter itself (0.68): So its evidence for support or rejection of a specific theoretical prediction almost vanished. The bitter truth is that simply nothing can be deduced from the data - a result that most principal investigators hate since it means wasted time and money. And that's where the educated cheating starts. Let's say the theoretical prediction of the critical exponent a_3 is 0.73 (remember that the data were generated with a true value of 0.63): Simply fix parameter a_3 to 0.73

```
modelFunction=a1*Abs[x-a2]^(-0.73);
parametersOfModelFunction={a1,a2};
```

```
startParameters={ {a1,1.9}, {a2,9.99}};
```

and fit parameters a_1 and a_2 only:

```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,  
modelFunction,argumentOfModelFunction,parametersOfModelFunction,  
CurveFitOptionStartParameters -> startParameters,  
CurveFitOptionConfidenceLevel ->  
confidenceLevelOfParameterErrors];  
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",  
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,  
curveFitInfo];
```

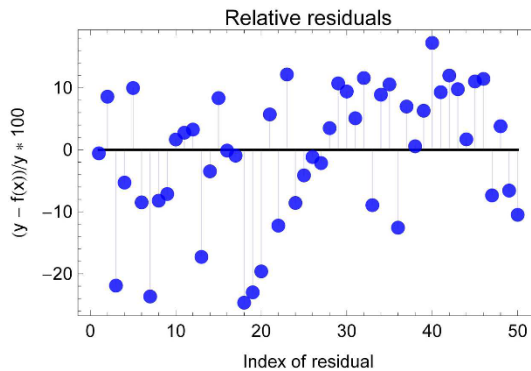
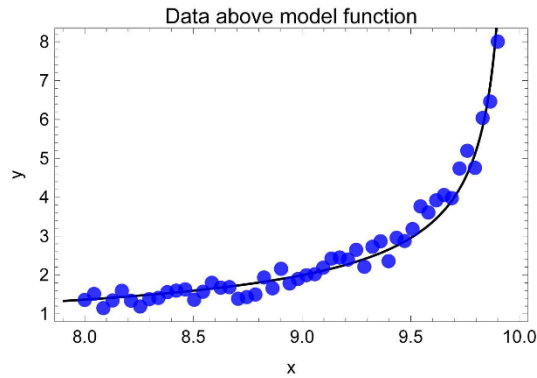


Standard deviation of fit = 1.772×10^{-1}
Reduced chi-square of fit = 8.445×10^{-1}

	Value	Standard error	Confidence region
Parameter a1	2.13617	0.0487269	{1.96538, 2.30696}
Parameter a2	10.0775	0.017931	{10.0146, 10.1403}

A very good looking fit is the result with a very convincing residuals plot which may easily be published to be *in perfect agreement with the theoretical prediction of 0.73*. But with about the same evidence it could be argued for a critical exponent a_3 of value 0.53:

```
modelFunction=a1*Abs[x-a2]^(-0.53);
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters,
CurveFitOptionConfidenceLevel->
confidenceLevelOfParameterErrors];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
curveFitInfo];
```



Standard deviation of fit = 2.008×10^{-1}

Reduced chi-square of fit = 1.083

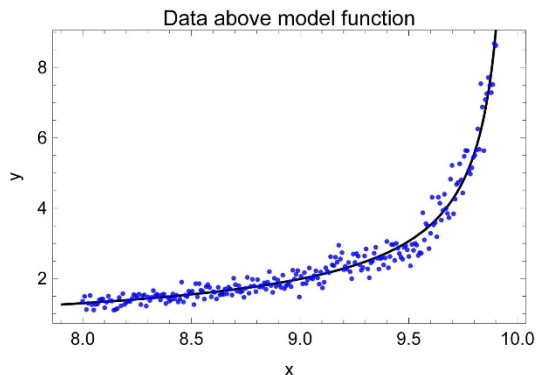
	Value	Standard error	Confidence region
Parameter a1 =	1.95121	0.0335399	{1.83365, 2.06877}
Parameter a2 =	9.95776	0.00900621	{9.92619, 9.98933}

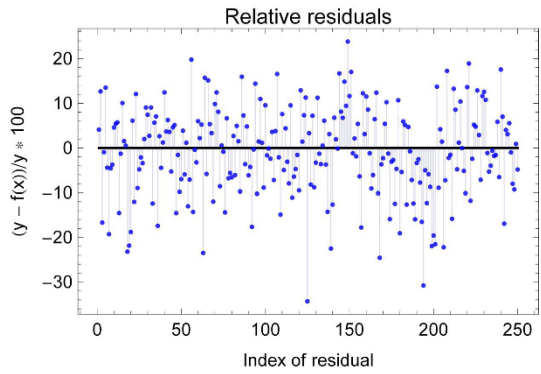
The fit again is convincing and *in perfect agreement with ...* The situation becomes only somewhat better if the number of data is increased. For a fivefold data boost

```
numberOfData=250;
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange,
CalculatedDataOptionErrorType -> errorType,
CalculatedDataOptionDistance -> argumentDistance];
```

and a fit with the complete 3-parameter model function

```
modelFunction=a1*Abs[x-a2]^(-a3);
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,1.9},{a2,9.99},{a3,-0.6}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionConfidenceLevel ->
confidenceLevelOfParameterErrors];
pointSize=0.01;
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
curveFitInfo,GraphicsOptionPointSize -> pointSize];
```

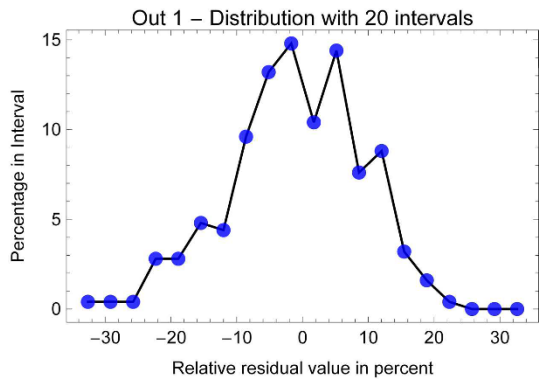




Standard deviation of fit = 1.92×10^{-1}
Reduced chi-square of fit = 9.885×10^{-1}

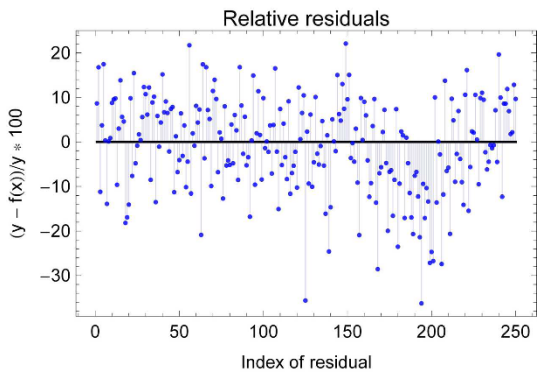
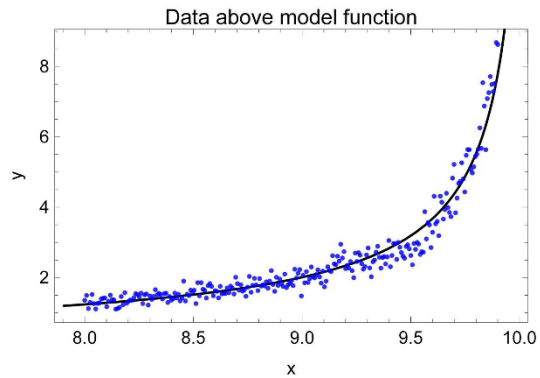
	Value	Standard error	Confidence region
Parameter a1	1.96714	0.0222953	{1.89289, 2.0414}
Parameter a2	9.97971	0.0120999	{9.93941, 10.02}
Parameter a3	0.602932	0.0193802	{0.538389, 0.667475}

```
numberOfIntervals=10;  
CIP `CurveFit `ShowFitResult[  
  {"RelativeResidualsDistribution"},xyErrorData,  
  curveFitInfo,NumberOfIntervalsOption -> numberOfIntervals];
```



the estimated value of the critical exponent a_3 improves and its confidence region inevitably shrinks. The distribution of the residuals looks like a distorted bell curve. But the evidence for both false theoretical predictions with values 0.73 and 0.53 would still be convincing: Theoretical prediction 0.73

```
modelFunction=a1*Abs[x-a2]^(-0.73);
parametersOfModelFunction={a1,a2};
startParameters={{a1,1.9},{a2,9.99}};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionConfidenceLevel ->
confidenceLevelOfParameterErrors];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
"SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
curveFitInfo,GraphicsOptionPointSize -> pointSize];
```

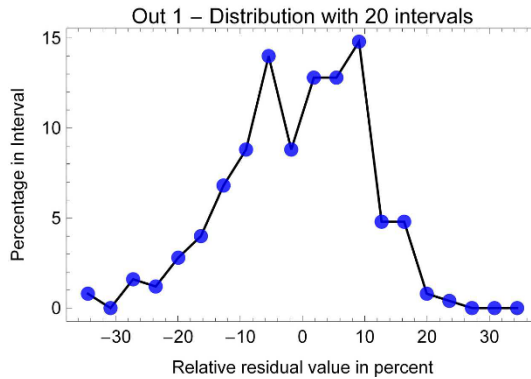


Standard deviation of fit = 2.028×10^{-1}

Reduced chi-square of fit = 1.103

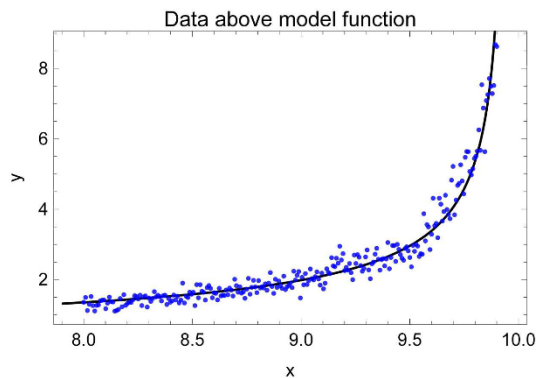
	Value	Standard error	Confidence region
Parameter a1	2.10871	0.0216214	{2.03671, 2.18072}
Parameter a2	10.067	0.00791856	{10.0406, 10.0934}

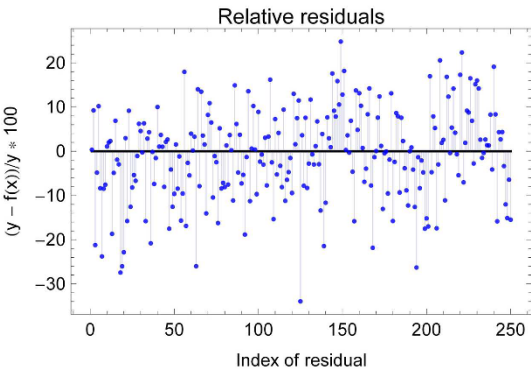
```
CIP `CurveFit `ShowFitResult[
  {"RelativeResidualsDistribution"},xyErrorData,
  curveFitInfo,NumberOfIntervalsOption -> numberOfIntervals];
```



looks approximately as good as the 3-parameter-fit and theoretical prediction 0.53:

```
modelFunction=a1*Abs[x-a2]^(-0.53);
curveFitInfo=CIP `CurveFit `FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters,
  CurveFitOptionConfidenceLevel ->
  confidenceLevelOfParameterErrors];
CIP `CurveFit `ShowFitResult[{"FunctionPlot", "RelativeResidualsPlot",
  "SDFit", "ReducedChiSquare", "ParameterErrors"},xyErrorData,
  curveFitInfo,GraphicsOptionPointSize -> pointSize];
```

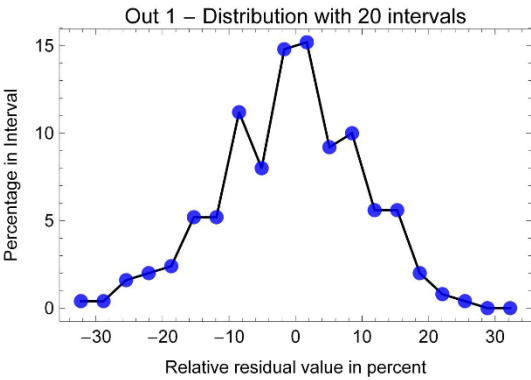




Standard deviation of fit = 1.991×10^{-1}
Reduced chi-square of fit = 1.063

	Value	Standard error	Confidence region
Parameter a1 =	1.92718	0.0147138	{1.87818, 1.97618}
Parameter a2 =	9.94512	0.00372884	{9.93271, 9.95754}

```
CIP `CurveFit `ShowFitResult[
  {"RelativeResidualsDistribution"},xyErrorData,
  curveFitInfo,NumberOfIntervalsOption -> numberOfIntervals];
```



So a lot more experimental data would be needed to really make clear decisions. With the aid of the `GetNumberOfData` method of the CIP `CurveFit` package the necessary number of data for a desired width of a parameters' confidence region may be estimated (see the previous section). For a desired confidence region width of 0.04 for parameter a_3

```
desiredWidthOfConfidenceRegion=0.04;
indexOfParameter=3;
```

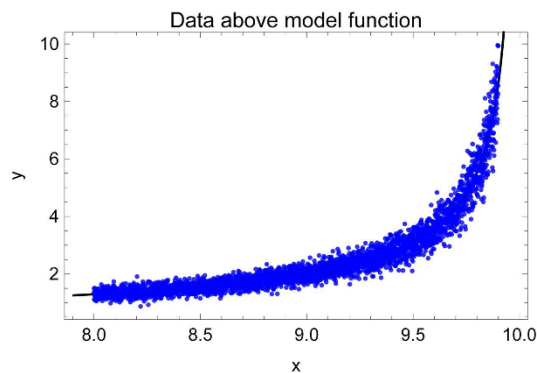
the number of data must be increased to

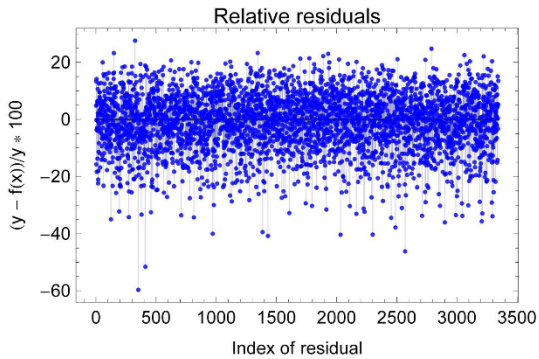
```
modelFunction=a1*Abs[x-a2]^(-a3);
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2,a3};
startParameters={a1,1.9},{a2,9.99},{a3,-0.6}};
numberOfData=CIP`CurveFit`GetNumberOfData[
  desiredWidthOfConfidenceRegion,indexOfParameter,
  pureOriginalFunction,argumentRange,standardDeviationRange,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters,
  CurveFitOptionConfidenceLevel ->
  confidenceLevelOfParameterErrors,
  CalculatedDataOptionErrorType -> errorType,
  CalculatedDataOptionDistance -> argumentDistance]
```

3344

The corresponding fit with this estimated number of data

```
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
  argumentRange,numberOfData,standardDeviationRange,
  CalculatedDataOptionErrorType -> errorType,
  CalculatedDataOptionDistance -> argumentDistance];
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,
  CurveFitOptionStartParameters -> startParameters,
  CurveFitOptionConfidenceLevel ->
  confidenceLevelOfParameterErrors];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","RelativeResidualsPlot",
  "SDFit","ReducedChiSquare","ParameterErrors"},xyErrorData,
  curveFitInfo,GraphicsOptionPointSize -> pointSize];
```

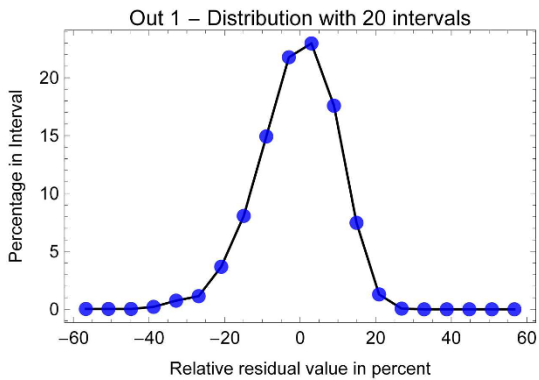




Standard deviation of fit = 1.912×10^{-1}
Reduced chi-square of fit = 9.798×10^{-1}

	Value	Standard error	Confidence region
Parameter a1 =	1.99724	0.00747252	{1.97263, 2.02185}
Parameter a2 =	9.99805	0.00417287	{9.98431, 10.0118}
Parameter a3 =	0.628188	0.00604781	{0.60827, 0.648106}

```
numberOfIntervals=30;  
CIP 'CurveFit' ShowFitResult[  
  {"RelativeResidualsDistribution", xyErrorData,  
   curveFitInfo, NumberOfIntervalsOption -> numberOfIntervals];
```



finally allows estimates within the required precision. For many experimental setups however the necessary increase of data would be completely out of reach due to restrictions in time and money. Therefore the sketched kind of educated cheating is unfortunately more widespread than it ought to be (and even worse is often

combined with an elimination of outliers after the fit: A "very successful strategy" to tune the data). In most cases experimentalists do not even have a bad conscience since the final plots look good. Therefore a clear trend can be detected for experimental data analysis to follow theoretical predictions (this can be superbly shown in the field of critical phenomena where the theoretical predictions changed over the decades and the experimental data analysis with them in close accordance). But it should not be forgotten that cheating simply has nothing to do with science - and in the end someone will detect it regardless how educated it was hidden.

2.5.5 Experimental errors and data transformation

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`CurveFit`
```

The errors σ_i of the y_i values do not only influence the errors of the parameters of the fitted model function but they also influence the parameters' values themselves. This is often ignored but obvious if it is remembered that curve fitting means minimization of $\chi^2(a_1, \dots, a_L)$:

$$\chi^2(a_1, \dots, a_L) = \sum_{i=1}^K \left(\frac{y_i - f(x_i, a_1, \dots, a_L)}{\sigma_i} \right)^2 \rightarrow \text{minimize!}$$

Since the errors σ_i are part of the sum of squares they contribute to the determination of the minimum location of $\chi^2(a_1, \dots, a_L)$. Only in the special case that all errors σ_i are equal

$$\sigma_i = \sigma$$

they are a mere factor σ that can be factored out of the sum and therefore does not influence the minimum. The influence of the errors σ_i can be illustrated with the following (artificial) example of twenty simulated data

```
numberOfData=20;
```

around the function

$$y = f(x) = 2e^{-\frac{1}{x}}$$

```
pureOriginalFunction=Function[x, 2.0*Exp[-1.0/x]];
```


in the argument range [1.0, 8.0]

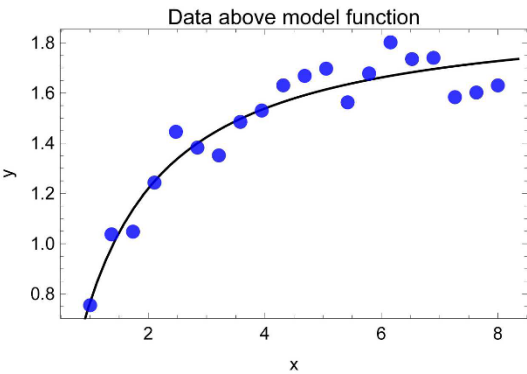
```
argumentRange={1.0,8.0};
```

with a relative standard deviation of 5%

```
standardDeviationRange={0.05,0.05};
errorType="Relative";
xyErrorData=CIP `CalculatedData `GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange,
CalculatedDataOptionErrorType -> errorType];
```

that are fitted with corrected estimates of parameters' errors for comparison purposes:

```
modelFunction=a1*Exp[-a2/x];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2};
varianceEstimator="ReducedChiSquare";
curveFitInfo=CIP `CurveFit `FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionVarianceEstimator -> varianceEstimator];
CIP `CurveFit `ShowFitResult[{"FunctionPlot","SDFit",
"ReducedChiSquare","ParameterErrors"},xyErrorData,curveFitInfo];
```



Standard deviation of fit = 7.272×10^{-2}

Reduced chi-square of fit = 1.122

	Value	Standard error	Confidence region
Parameter a1	= 1.93972	0.0396424	{1.89894, 1.98049}
Parameter a2	= 0.92742	0.051963	{0.873972, 0.980868}

If the errors σ_i are asymmetrically enlarged by different factors from 10.0 (ten-fold increase) to 1.0 (no change)

```

minFactor=1.0;
maxFactor=10.0;
errorTransformationFactors=Table[{i,{i,maxFactor,minFactor,
-(maxFactor-minFactor)/(Length[xyErrorData]-1)}}];
newXyErrorData=Table[{xyErrorData[[i,1]],xyErrorData[[i,2]],
xyErrorData[[i,3]]*errorTransformationFactors[[i]]},
{i,Length[xyErrorData]}}];

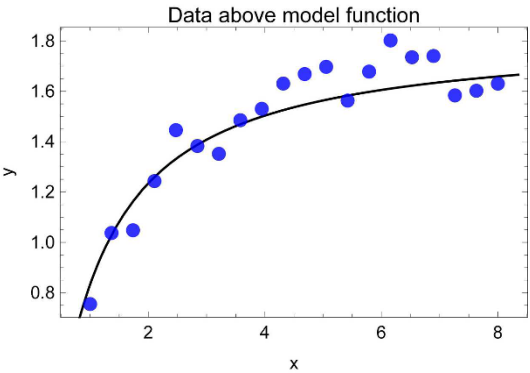
```

the estimated optimum values of the parameters become clearly different:

```

curveFitInfo=CIP`CurveFit`FitModelFunction[newXyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionVarianceEstimator->varianceEstimator];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","SDFit",
"ReducedChiSquare","ParameterErrors"},xyErrorData,curveFitInfo];

```



Standard deviation of fit = 8.842×10^{-2}
 Reduced chi-square of fit = 1.659

	Value	Standard error	Confidence region
Parameter a1 =	1.83037	0.0368555	{1.79246, 1.86828}
Parameter a2 =	0.787937	0.107727	{0.67713, 0.898743}

The parameter estimates changed by around 5-10% of their absolute values although the x_i and y_i values were not changed at all. Also the values of corrected parameters' errors increased due to the increase of the data's errors.

As far as the popular data transformations are considered the outlined context may play a more or less pronounced role. It is still common in lab data analysis to linearize model functions to a straight line if possible (despite the existence of non-linear curve fitting software). For the model function above linearization may be easily performed by simple application of the natural logarithm

$$y = f(x) = a_1 e^{-\frac{a_2}{x}}$$

$$\ln(y) = \ln\left(a_1 e^{-\frac{a_2}{x}}\right) = \ln a_1 - a_2 \frac{1}{x}$$

which results in a straight line

$$y = f(x) = a_1 + a_2 x$$

with the necessary non-linear data transformations:

$$x_i \rightarrow \frac{1}{x_i} ; y_i \rightarrow \ln(y_i)$$

If data are transformed it is often forgotten that the errors σ_i must be transformed too according to standard error propagation:

$$\sigma_i \rightarrow \sqrt{\left(\frac{\partial \ln(y_i)}{\partial y_i}\right)^2 \sigma_i^2} = \left(\frac{\partial \ln(y_i)}{\partial y_i}\right) \sigma_i = \frac{\sigma_i}{y_i}$$

Note that the neglect of this error transformation is perhaps the second most frequent mistake in lab data analysis. (The most frequent mistake is the lab journal's report of a mean in combination with the standard deviation of a single measurement and not the correct standard deviation of the mean.) In summary each data triple of the xy-error data must be transformed as follows:

$$(x_i, y_i, \sigma_i) \rightarrow \left(\frac{1}{x_i}, \ln(y_i), \frac{\sigma_i}{y_i}\right)$$

Standard error propagation assumes vanishingly small errors since it belongs to linear statistics (with Taylor series expansions up to the first derivative only). Therefore the transformed errors and the original errors do only correspond in an approximate manner. This may have more or less influence on the estimated values of the parameters after linearization depending on the specific fit problem.

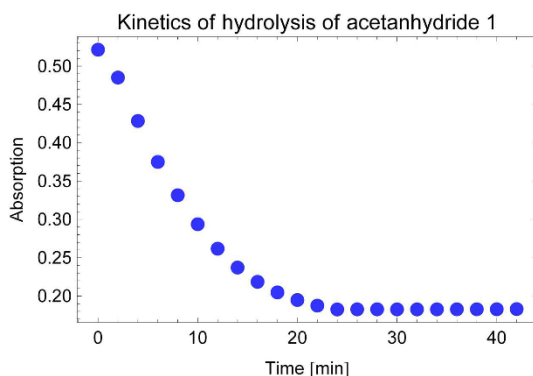
2.6 Empirical enhancement of theoretical model functions

```
Clear["Global`*"];
<<CIP`ExperimentalData`
<<CIP`DataTransformation`
<<CIP`Graphics`
<<CIP`CurveFit`
```

Suppose there is a well-defined theoretical model function but the x and y quantities it associates can not be measured directly. Preprocessing steps are necessary to construct the data in question which may introduce systematic errors. An example

is outlined in Appendix A that shows the extraction of kinetics data for a chemical reaction (in this case the hydrolysis of acetanhydride) from time dependent infrared (IR) spectra: There are two different methods advised to extract the data: One straight forward method denoted 1 and one more elaborate method denoted 2. The results are provided by the CIP ExperimentalData package. The data produced by method 1 are as follows:

```
xyData=
  CIP`ExperimentalData`GetAcetanhydrideKineticsData1[
  ];
errorValue=1.0;
xyErrorData=CIP`DataTransformation`AddErrorToXYData[xyData,
  errorValue];
labels={"Time [min]", "Absorption",
  "Kinetics of hydrolysis of acetanhydride 1"};
CIP`Graphics`PlotXyErrorData[xyErrorData, labels]
```



Note that a standard weight of 1.0 was added as an error to the xy data to obtain xy-error data since the preprocessing method did not yield any error estimate. All estimates for parameters' errors thus need a correction deduced from χ^2_{red} (see previous sections).

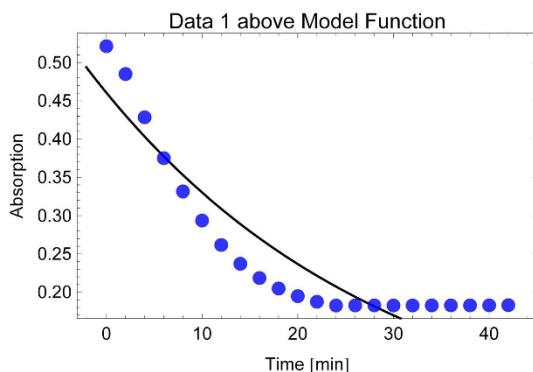
The hydrolysis of acetanhydride in water is a reaction of (pseudo) first-order which is theoretically described by a simple exponential decay:

$$y = f(x) = a_1 e^{-a_2 x}$$

But a direct fit of this model function

```
modelFunction=a1*Exp[-a2*x];
argumentOfModelFunction=x;
parametersOfModelFunction={a1,a2};
varianceEstimator="ReducedChiSquare";
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
```

```
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionVarianceEstimator -> varianceEstimator];
labels={"Time [min]","Absorption","Data 1 above Model Function"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot"},xyErrorData,
curveFitInfo,CurveFitOptionLabels -> labels];
```



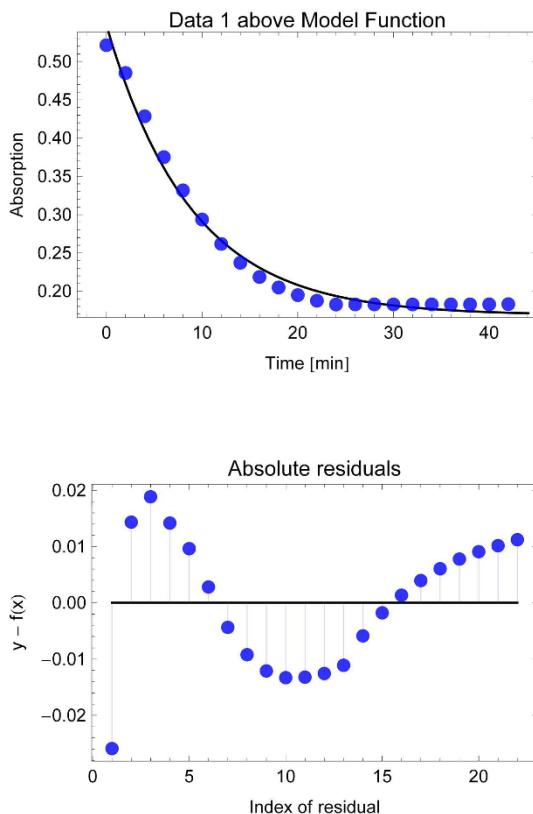
fails completely. Due to preprocessing method 1 the data do not direct to a zero absorption with increasing time (acetanhydride vanishes with reaction progress) but to a constant value above zero (a so called background caused by the extraction process, see Appendix A). Therefore the theoretical model function must be enhanced by (at least) an empirical constant background parameter a_3 that takes this deficiency into account:

$$y = f(x) = a_1 e^{-a_2 x} + a_3$$

```
modelFunction=a1*Exp[-a2*x]+a3;
parametersOfModelFunction={a1,a2,a3};
startParameters={{a1,0.4},{a2,0.1},{a3,0.2}};
```

The enhanced fit

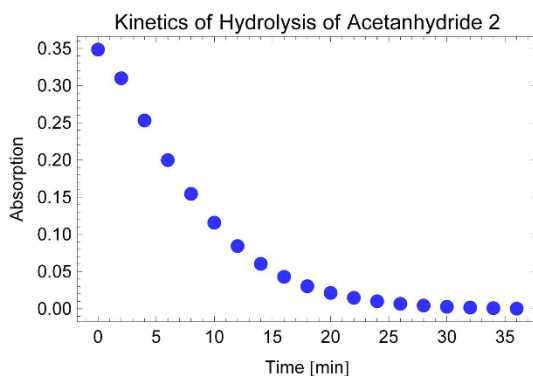
```
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters -> startParameters,
CurveFitOptionVarianceEstimator -> varianceEstimator];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ParameterErrors"},xyErrorData,curveFitInfo,
CurveFitOptionLabels -> labels];
```



	Value	Standard error	Confidence region
Parameter a1 =	0.378912	0.00979157	{0.368856, 0.388968}
Parameter a2 =	0.112982	0.00687499	{0.105921, 0.120043}
Parameter a3 =	0.168445	0.0051983	{0.163106, 0.173783}

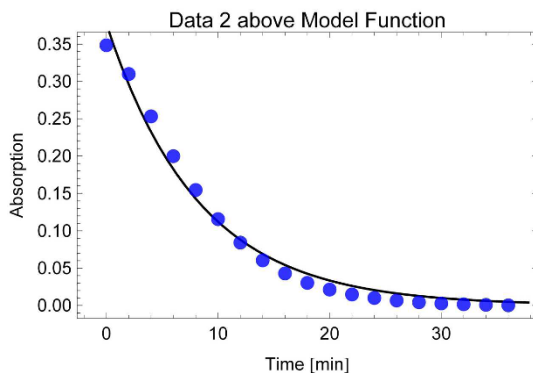
leads to an improved description of the data but reveals a strong systematic deviation pattern of the residuals. In contrast to method 1 the more elaborate preprocessing method 2 tries to estimate the background contribution in advance (see details in Appendix A):

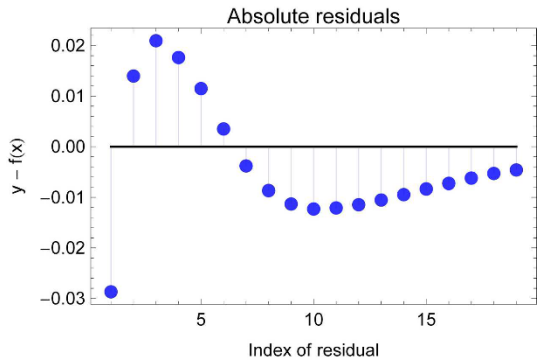
```
xyData=
  CIP\ExperimentalData\GetAcetanhydrideKineticsData2[
  ];
errorValue=1.0;
xyErrorData=CIP\DataTransformation\AddErrorToXYData[xyData,
  errorValue];
labels={"Time [min]", "Absorption",
  "Kinetics of Hydrolysis of Acetanhydride 2"};
CIP\Graphics\PlotXyErrorData[xyErrorData, labels]
```



Now the absorption values seem to direct to zero. But a direct fit of the pure theoretical model

```
modelFunction=a1*Exp[-a2*x];
parametersOfModelFunction={a1,a2};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionVarianceEstimator->varianceEstimator];
labels={"Time [min]","Absorption","Data 2 above Model Function"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ParameterErrors"},xyErrorData,curveFitInfo,
CurveFitOptionLabels->labels];
```

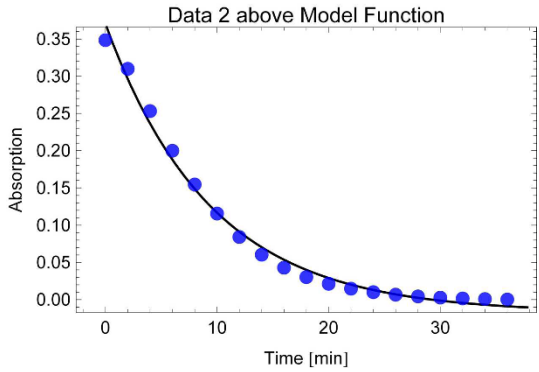


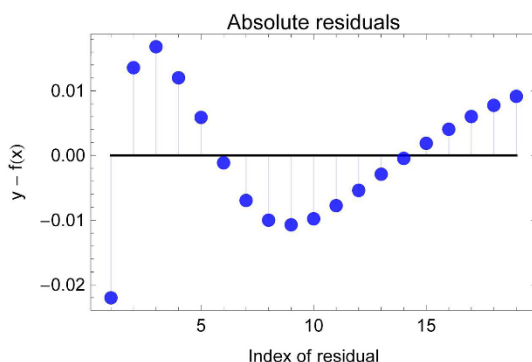


	Value	Standard error	Confidence region
Parameter a1 =	0.377157	0.0104281	{0.366413, 0.387901}
Parameter a2 =	0.121256	0.00534371	{0.11575, 0.126761}

still suggests the use of an additional constant background parameter a_3

```
modelFunction=a1*Exp[-a2*x]+a3;  
parametersOfModelFunction={a1,a2,a3};  
startParameters={{a1,0.4},{a2,0.1},{a3,0.2}};  
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,  
  modelFunction,argumentOfModelFunction,parametersOfModelFunction,  
  CurveFitOptionStartParameters -> startParameters,  
  CurveFitOptionVarianceEstimator -> varianceEstimator];  
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",  
  "ParameterErrors"},xyErrorData,curveFitInfo,  
  CurveFitOptionLabels -> labels];
```

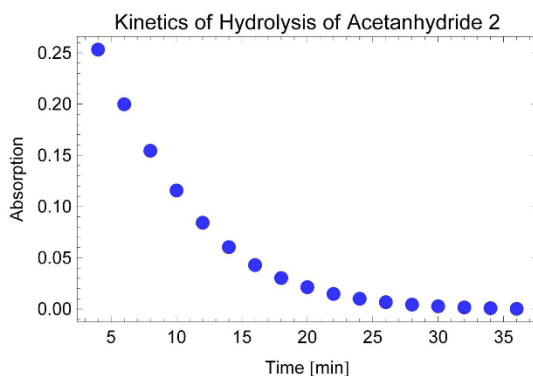




	Value	Standard error	Confidence region
Parameter a1 =	0.38797	0.00863007	{0.379061, 0.396878}
Parameter a2 =	0.106077	0.00623914	{0.0996363, 0.112517}
Parameter a3 =	-0.0174831	0.00606562	{-0.0237445, -0.0112218}

which results in a estimated value for a_3 that is at least close to zero (so the background correction of the more elaborate preprocessing method was not in vain). The visual inspection of the data also suggests to treat the first two points as outliers: If they are removed from the xy-error data

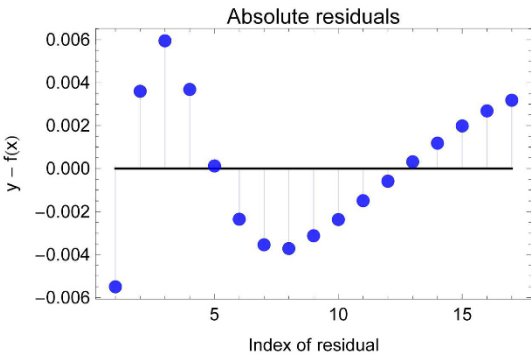
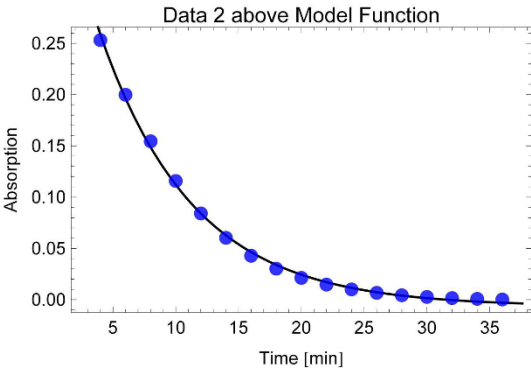
```
xyErrorData=Drop[xyErrorData,2];
labels={"Time [min]","Absorption",
"Kinetics of Hydrolysis of Acetanhydride 2"};
CIP`Graphics`PlotXyErrorData[xyErrorData,labels]
```



the remaining data do not only look better but lead to an improved fit

```
modelFunction=a1*Exp[-a2*x]+a3;
parametersOfModelFunction={a1,a2,a3};
```

```
startParameters={{a1,0.4},{a2,0.1},{a3,0.2}};
curveFitInfo=CIP'CurveFit'FitModelFunction[xyErrorData,
modelFunction,argumentOfModelFunction,parametersOfModelFunction,
CurveFitOptionStartParameters->startParameters,
CurveFitOptionVarianceEstimator->varianceEstimator];
labels={"Time [min]","Absorption","Data 2 above Model Function"};
CIP'CurveFit'ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ParameterErrors"},xyErrorData,curveFitInfo,
CurveFitOptionLabels->labels];
```



	Value	Standard error	Confidence region
Parameter	a1 = 0.453659	0.00879298	{0.444541, 0.462778}
Parameter	a2 = 0.134134	0.00371886	{0.130277, 0.13799}
Parameter	a3 = -0.00662982	0.00180625	{-0.00850296, -0.00475667}

with significantly smaller residuals and a further decreased background parameter a_3 . There is still a clear systematic deviation pattern of the residuals but this is probably the best we can get. Always keep in mind that the introduction of new empirical parameters and the removal of apparent outliers are dangerous procedures that are an ideal basis for educated cheating: In the end you can obtain (nearly) any

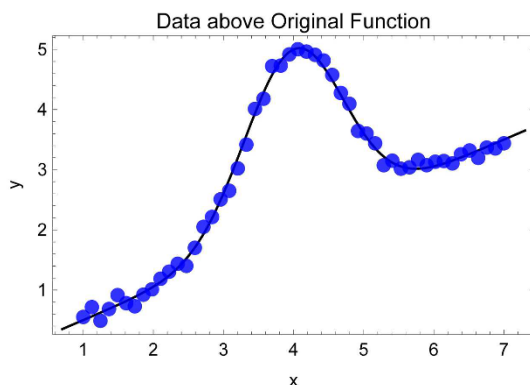
result you like to get and this is not the objective of science which claims to describe the real world out there. But careful and considerate use of these procedures may extract information from data that would otherwise be lost.

2.7 Data smoothing with cubic splines

```
Clear["Global`*"];
<<CIP`CalculatedData`
<<CIP`Graphics`
<<CIP`CurveFit`
<<CIP`ExperimentalData`
<<FunctionApproximations`
```

Data smoothing with cubic splines is controlled by the specified χ_{red}^2 value (see above). Depending on the χ_{red}^2 value there are two smoothing extremes: A small χ_{red}^2 value enforces small residuals and restricts the smoothing function to close proximity of the data points whereas a high χ_{red}^2 value allows for larger residuals but forces the curvature of the smoothing function to minimize towards a straight line. This may be demonstrated with fifty simulated xy-error data around the Gaussian-peak shaped function:

```
numberOfData=50;
pureOriginalFunction=Function[x,0.5*x+3.0*Exp[-(x-4.0)^2]];
argumentRange={1.0,7.0};
standardDeviationRange={0.1,0.1};
xyErrorData=CIP`CalculatedData`GetXyErrorData[pureOriginalFunction,
argumentRange,numberOfData,standardDeviationRange];
labels={"x","y","Data above Original Function"};
CIP`Graphics`PlotXyErrorDataAboveFunction[xyErrorData
,pureOriginalFunction,labels]
```

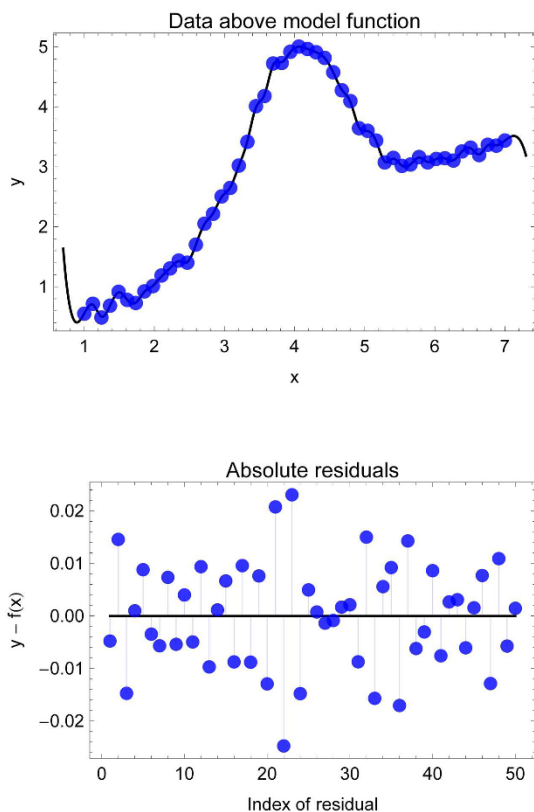


A small χ_{red}^2 value of 0.01

```
reducedChiSquare=0.01;
```

leads to mere interpolation between the data without smoothing

```
curveFitInfo=
  CIP 'CurveFit' 'FitCubicSplines[xyErrorData,reducedChiSquare];
  CIP 'CurveFit' 'ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
    "ReducedChiSquare","CorrelationCoefficient"},xyErrorData,
    curveFitInfo];
```



Reduced chi-square of fit = $1. \times 10^{-2}$

Out 1 : Correlation coefficient = 0.999973

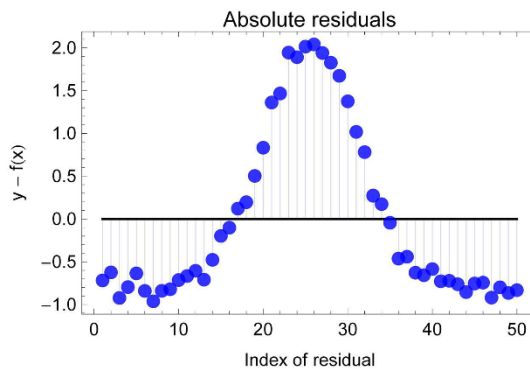
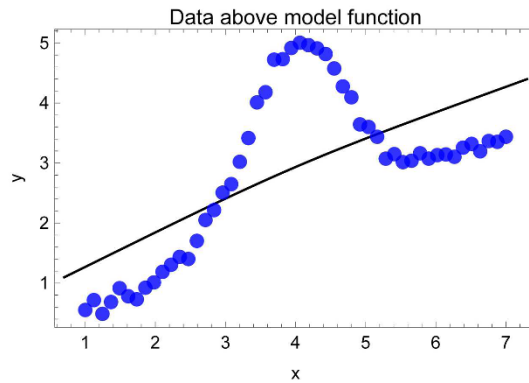
and small residuals. Note that the correlation coefficient that indicates the agreement of data and machine output is (almost) one which means a perfect correlation:

Since the data are erroneous this outcome indicates a so called overfitting of the data (which is to be avoided for convincing smoothing). A high χ_{red}^2 value of 100

```
reducedChiSquare=100.0;
```

leads to a straight line

```
curveFitInfo=CIP`CurveFit`FitCubicSplines[xyErrorData,
reducedChiSquare];
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ReducedChiSquare","CorrelationCoefficient"},xyErrorData,
curveFitInfo];
```

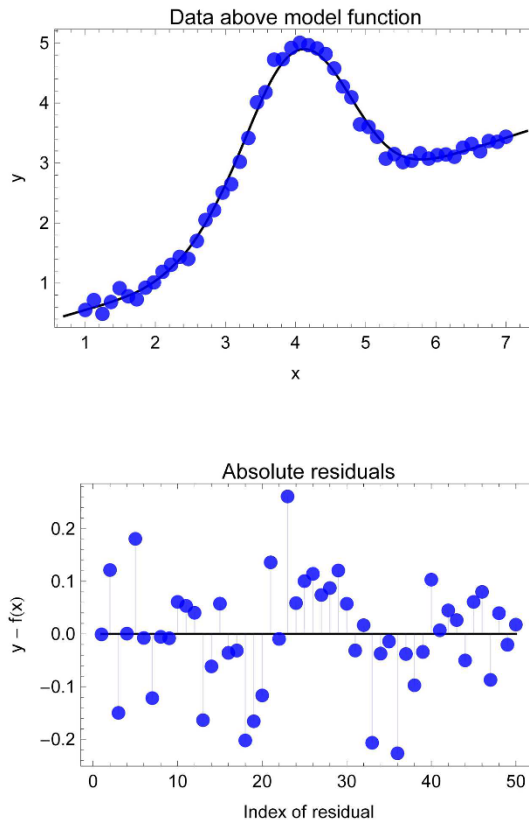


Reduced chi-square of fit = $1. \times 10^2$

Out 1 : Correlation coefficient = 0.683152

without adequate data description and a small correlation coefficient (which is as unfavorable as a perfect correlation for erroneous data). In practice a χ_{red}^2 value is initially chosen that is around 1 to produce a smooth and balancing model function with a convincing residuals plot

```
reducedChiSquare=1.0;
curveFitInfo=CIP'CurveFit'FitCubicSplines[xyErrorData,
reducedChiSquare];
CIP'CurveFit'ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ReducedChiSquare","CorrelationCoefficient"},xyErrorData,
curveFitInfo];
```



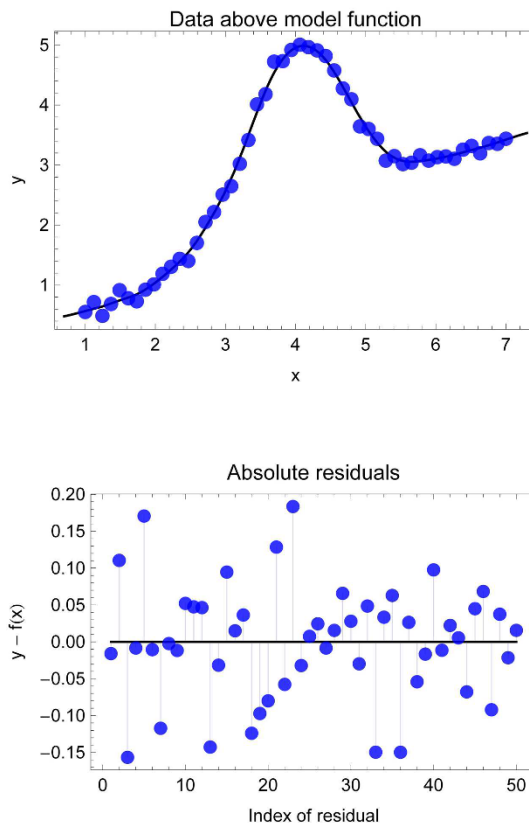
Reduced chi-square of fit = 1.001

Out 1 : Correlation coefficient = 0.997407

and a reasonable high correlation coefficient. Since the smoothing cubic splines procedure tries to minimize the overall curvature over the whole argument range

the curved peak region of the current example is comparatively poorly described: A systematic deviation pattern of positive residuals is visible in this middle region of the residuals plot. In this case the χ^2_{red} value should be lowered which enforces smaller residuals to describe the peak region more precisely:

```
reducedChiSquare=0.6;
curveFitInfo=CIP 'CurveFit 'FitCubicSplines[xyErrorData,
reducedChiSquare];
CIP 'CurveFit 'ShowFitResult[{"FunctionPlot", "AbsoluteResidualsPlot",
"ReducedChiSquare", "CorrelationCoefficient"}, xyErrorData,
curveFitInfo];
```



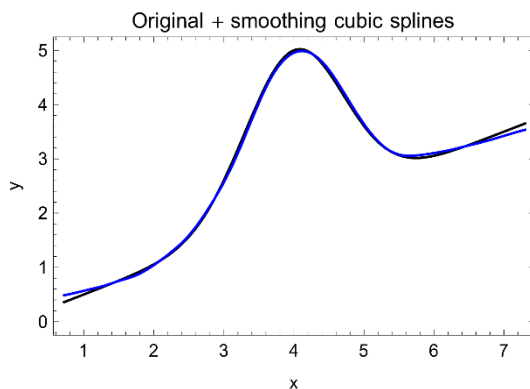
Reduced chi-square of fit = $6. \times 10^{-1}$

Out 1 : Correlation coefficient = 0.998401

This results in an overall acceptable fit. Note that the correlation coefficient is not too valuable for a goodness-of-smoothing discussion since a higher value does

not imply better smoothing due to the increased tendency towards overfitting. The smoothing model function may be finally compared (overlayed) with the original Gaussian-peak shaped function that was used for the simulated data generation

```
pureSmoothingFunction=Function[x,CalculateFunctionValue[x,
curveFitInfo]];
pureFunctions={pureOriginalFunction,pureSmoothingFunction};
plotRange={0.0,5.0};
plotStyle={{Thickness[0.005],Black},{Thickness[0.005],Blue}};
labels={"x","y","Original + smoothing cubic splines"};
CIP`Graphics`Plot2dFunctions[pureFunctions,argumentRange,plotRange,
plotStyle,labels]
```



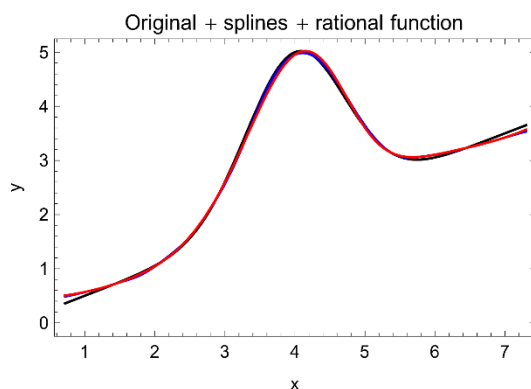
to demonstrate their close proximity and thus a successful model approximation by mere data smoothing. The cubic splines based smoothing model function may be used for interpolating calculations of function values and derivatives. Calculations outside the data's argument range are possible but useless since the cubic splines may have an arbitrary value there: As already mentioned reasonable extrapolations are in principle out of reach if the structural form of the model function is not known. For publishing purposes the internal representation of the smoothing model function is somewhat lengthy: For each (x_i, y_i, σ_i) data triple of the xy-error data a cubic polynomial with 4 parameters is constructed so that the 50 data triples require 200 parameters for the cubic splines. To achieve a more condensed representation the smoothing function may be approximated by a rational function which is constructed by mere trial and error (in this case with a numerator of order 8 and a denominator of order 4):

```
rationalFunction=FunctionApproximations`RationalInterpolation[
CalculateFunctionValue[x,curveFitInfo],{x,8,4},
{x,argumentRange[[1]],argumentRange[[2]]}]
```


$$\frac{0.492884 - 0.61194x + 0.646898x^2 - 0.467755x^3 + 0.204442x^4 - 0.0524233x^5 + 0.00768635x^6 - 0.000597131x^7 + 0.0000192755x^8}{1 - 0.866354x + 0.295339x^2 - 0.0462791x^3 + 0.0028032x^4}$$

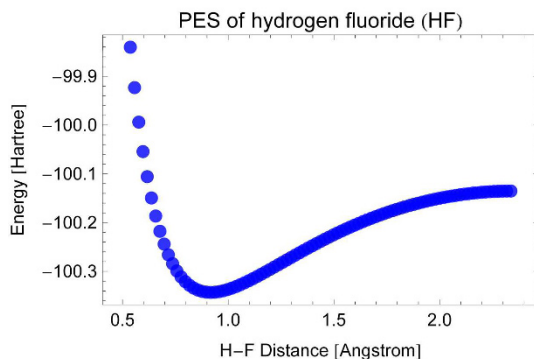
This condensed representation

```
pureRationalFunction=Function[argument,
  rationalFunction/.x -> argument];
pureFunctions={pureOriginalFunction,pureSmoothingFunction,
  pureRationalFunction};
plotStyle={{Thickness[0.005],Black},{Thickness[0.005],Blue},
  {Thickness[0.005],Red}};
labels={"x","y","Original + splines + rational function"};
CIP`Graphics`Plot2dFunctions[pureFunctions,argumentRange,plotRange,
  plotStyle,labels]
```



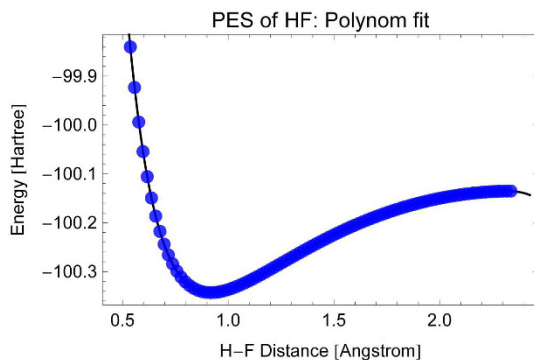
is of sufficient precision as shown by the final overlay. Another application of data smoothing is the representation of calculated data. This sounds absurd since calculated data can be calculated so there seems to be no need for data smoothing. But data calculation may be computationally very expensive in many cases. For example ab-initio quantum-chemical calculations for molecular properties are very time-consuming and therefore require a considerable percentage of the world's overall available computational power. If for example the potential energy surface (PES) of the diatomic molecule hydrogen fluoride is to be described the Schrodinger equation has to be solved for every desired distance between hydrogen and fluoride: Every single calculation may take from seconds up to minutes or hours depending on the level of approximation. The CIP ExperimentalData package contains a set of high precision single point calculations for hydrogen fluoride (see Appendix A):

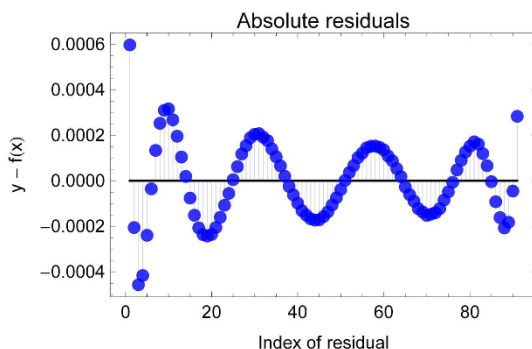
```
xyErrorData=CIP`ExperimentalData`GetHydrogenFluoridePESXyErrorData[];
labels={"H-F Distance [Angstrom]","Energy [Hartree]",
  "PES of hydrogen fluoride (HF)"};
CIP`Graphics`PlotXyErrorData[xyErrorData,labels]
```



The data are reported with a very small absolute error of 10^{-6} . So a very precise model function that is very near a pure interpolating function is in need. The standard approach with high-degree polynomials

```
modelFunction=
  A0+A1*R+A2*R^2+A3*R^3+A4*R^4+A5*R^5+A6*R^6+A7*R^7+A8*R^8+A9*R^9;
argumentOfModelFunction=R;
parametersOfModelFunction={A0,A1,A2,A3,A4,A5,A6,A7,A8,A9};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction];
labels={"H-F Distance [Angstrom]","Energy [Hartree]",
  "PES of HF: Polynom fit"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
  "ReducedChiSquare","AbsoluteResidualsStatistics",
  "CorrelationCoefficient"},xyErrorData,curveFitInfo,
  CurveFitOptionLabels -> labels];
```





Reduced chi-square of fit = 3.262×10^4

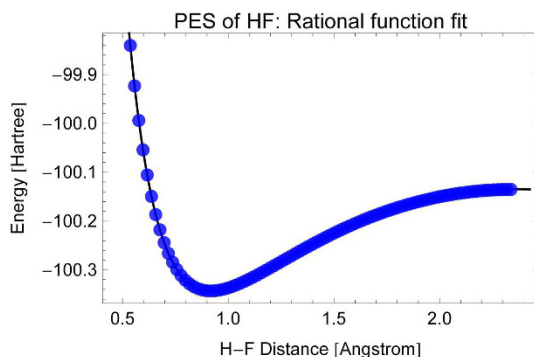
Definition of 'Residual (absolute)': Data - Model

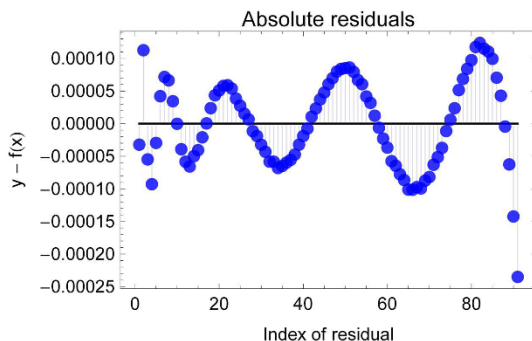
Out 1 : Residual (absolute): Mean/Median/Maximum Value = 1.4×10^{-4} / 1.35×10^{-4} / 5.97×10^{-4}

Out 1 : Correlation coefficient = 0.999998

leads to the well-known systematic oscillations (compare above) around the data that are beyond the required precision. A rational function fit is a little better

```
modelFunction=
  A0+A1*R^-1+A2*R^-2+A3*R^-3+A4*R^-4+A5*R^-5+A6*R^-6+A7*R^-7+
  A8*R^-8+A9*R^-9;
argumentOfModelFunction=R;
parametersOfModelFunction={A0,A1,A2,A3,A4,A5,A6,A7,A8,A9};
curveFitInfo=CIP`CurveFit`FitModelFunction[xyErrorData,
  modelFunction,argumentOfModelFunction,parametersOfModelFunction];
labels={"H-F Distance [Angstrom]","Energy [Hartree]",
  "PES of HF: Rational function fit"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
  "ReducedChiSquare","AbsoluteResidualsStatistics",
  "CorrelationCoefficient"},xyErrorData,curveFitInfo,
  CurveFitOptionLabels -> labels];
```





Reduced chi-square of fit = 5.16×10^3

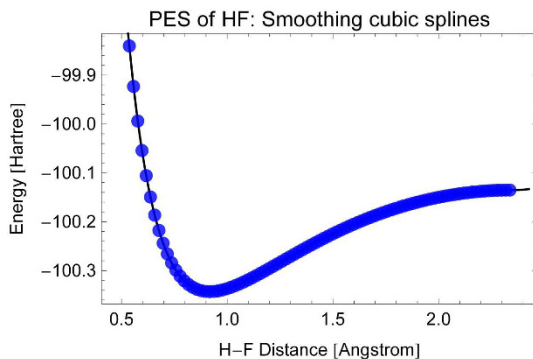
Definition of 'Residual (absolute)': Data - Model

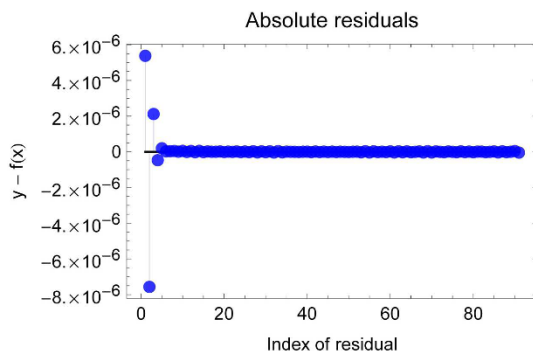
Out 1 : Residual (absolute): Mean/Median/Maximum Value = 5.69×10^{-5} / 5.55×10^{-5} / 2.35×10^{-4}

Out 1 : Correlation coefficient = 1.

but also beyond acceptability. Data smoothing with cubic splines and a χ^2_{red} value of 1 however

```
reducedChiSquare=1.0;
curveFitInfo=CIP`CurveFit`FitCubicSplines[xyErrorData,
reducedChiSquare];
labels={"H-F Distance [Angstrom]","Energy [Hartree]",
"PES of HF: Smoothing cubic splines"};
CIP`CurveFit`ShowFitResult[{"FunctionPlot","AbsoluteResidualsPlot",
"ReducedChiSquare","AbsoluteResidualsStatistics",
"CorrelationCoefficient"},xyErrorData,curveFitInfo,
CurveFitOptionLabels -> labels];
```





Reduced chi-square of fit = 1.

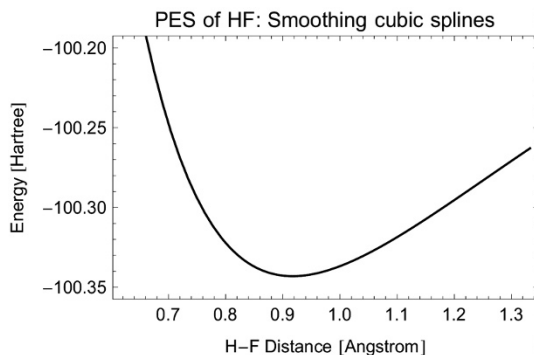
Definition of 'Residual (absolute)': Data - Model

Out 1 : Residual (absolute): Mean/Median/Maximum Value = 1.92×10^{-7} / 1.99×10^{-8} / 7.57×10^{-6}

Out 1 : Correlation coefficient = 1.

achieves an acceptable interpolation with residuals well within the required order of magnitude: Deviations are only more pronounced in the divergence region at small interatomic distances. Also note that a correlation coefficient of effectively one does not indicate overfitting in this situation since the data errors are very small. A check of the smoothing model function is a calculation of the minimum energy distance between hydrogen and fluoride that is known to be 0.917 Angstrom:

```
argumentRange={0.65,1.3};
functionValueRange={-100.35,-100.2};
CIP`Graphics`Plot2dFunction[Function[arg,CalculateFunctionValue[arg,
  curveFitInfo]],argumentRange,functionValueRange,labels]
```



```
FindMinimum[CIP`CurveFit`CalculateFunctionValue[x, curveFitInfo],
{x, 0.5, 1.5}]
```

```
{-100.343, {x → 0.917413}}
```

This correct result together with a vanishing derivative value at the minimum (which ought to be 0)

```
CIP`CurveFit`CalculateDerivativeValue[1, 0.917, curveFitInfo]
```

```
-0.000933712
```

assures an overall satisfactory model function that may be successfully used for interpolation purposes.

2.8 Cookbook recipes for curve fitting

As demonstrated in the previous sections curve fitting can be a challenging task. In this last section some cookbook recipes for curve fitting and data smoothing summarize different aspects outlined above.

- **The data:** Start with a thorough (visual) inspection of the data to avoid the GIGO (garbage-in/garbage-out) effect. Data analysis is not magic, it can not extract information out of nothing. Are the data reasonably scaled and distributed? Are the reported errors convincing? If no errors are available apply the standard weight 1.0 and correct errors with χ_{red}^2 . There are additional subtle problems with data that contain outliers, i.e. single data points with extraordinarily large errors. Outliers usually indicate experimental failure. If outliers can be easily detected they should always be removed from the data since they tend to mask themselves in a fitting procedure (they draw the model towards them to become invisible). Data which are known to be prone to contain outliers may deserve a completely different statistical treatment like the so called robust estimation which is beyond this introduction (see [Hampel 1986] or [Rousseeuw 2003] for further reading).
- **The model function:** Is a well-defined model function available? Does the number of data well exceed the number of parameters? Then go on. If no model function is known it might be worth to try to construct one by educated trial and error: This is quite often successful. Avoid model functions with redundant or highly similar parameters. If an educated guess seems to be unfeasible try data smoothing.
- **Linear or non-linear model function:** Is the model function linear in its parameters? Then the fit will work without further considerations. If not: Are parameters' start values approximately known in advance? Then try these values for local minimization. Otherwise an extensive start values search may be advised.

Don't give up too early if things are difficult. The parameters' start values are often the most difficult part of the game.

- **Problems with the fitting procedure:** If the fitting procedure crashes try to use an alternative minimization algorithm. If nothing helps there seems to be a severe problem with the model function or the parameters' start values. Do you use professional curve fitting software? A lot of programs do use (too) simple algorithms without appropriate safeguards that fail needlessly.
- **Goodness of fit:** Are the fitted parameters' values reasonable? Otherwise the minimization procedure sent you somewhere over the rainbow. Is the data plot above the fitted model function convincing, i.e. smooth and balancing? If not the fit failed. Is the residuals plot well within experimental errors and free of systematic deviation patterns? Then probably everything worked well. Other goodness of fit quantities may be used to support your assessment.
- **Parameter errors:** Is the χ^2_{red} value close to 1? If not the reported experimental errors are poor and should be corrected. Do you need high confidence? Then adjust the parameters' confidence level in accordance. Are the parameters' errors too large to make any decisions? Try to avoid strategies of educated cheating unless your career or PhD is in danger (then you should at least provide a convincing residuals plot because this is what most reviewers believe in).
- **Data transformation for linearization:** If possible simply avoid it and use non-linear curve fitting software. Final diagrams may then be linearized for your audience.
- **Data smoothing:** Adjust the set screws until you like the result with a convincing residuals plot. Then apply the smoothing model for interpolation (but never for extrapolation) purposes.

This chapter sketched general curve fitting issues with a broad range of applications. For many specific curve fitting tasks elaborate specific solutions already exist that avoid problems outlined in the previous sections. Thus the scientific literature should always be consulted in advance (which is of course a mandatory and sensible advice for all scientific endeavours to avoid a reinvention of the wheel).

<http://www.springer.com/978-3-319-32544-6>

From Curve Fitting to Machine Learning
An Illustrative Guide to Scientific Data Analysis and
Computational Intelligence

Zielesny, A.

2016, XV, 498 p. 343 illus., 200 illus. in color.,

Hardcover

ISBN: 978-3-319-32544-6