Curve Fitting

Introduction to Numerical Problem Solving, Spring 2017

Helsinki University of Applied Sciences

Motivation

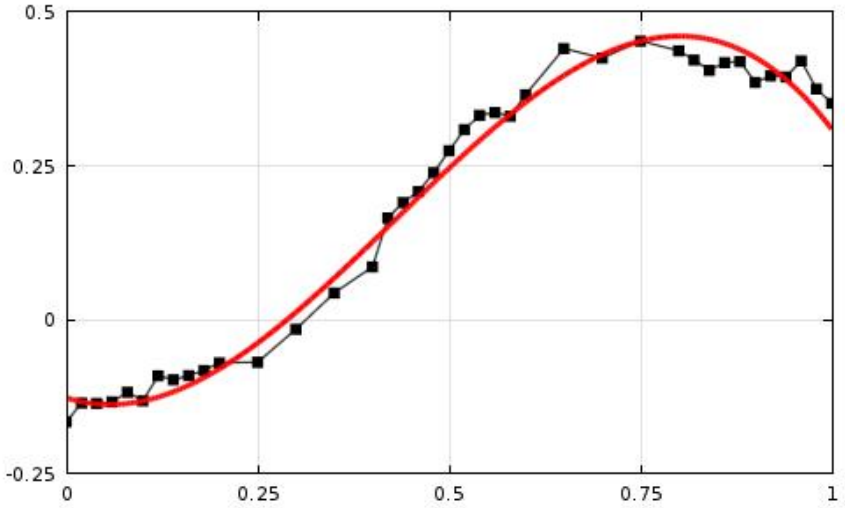
Given the n + 1 data points (x_i , y_i) i = 0, 1, 2, …, n

estimate y(x).

Motivation

If you have discrete data points (black dots) and you want to know the values between, you have two options:

- Curve fitting – you make a mathematical model that models the data as good as possible (red line)
- Interpolation – you interpolate the values between the points using some algorithm (black line)



Example

(a) Least-squares linear regressions

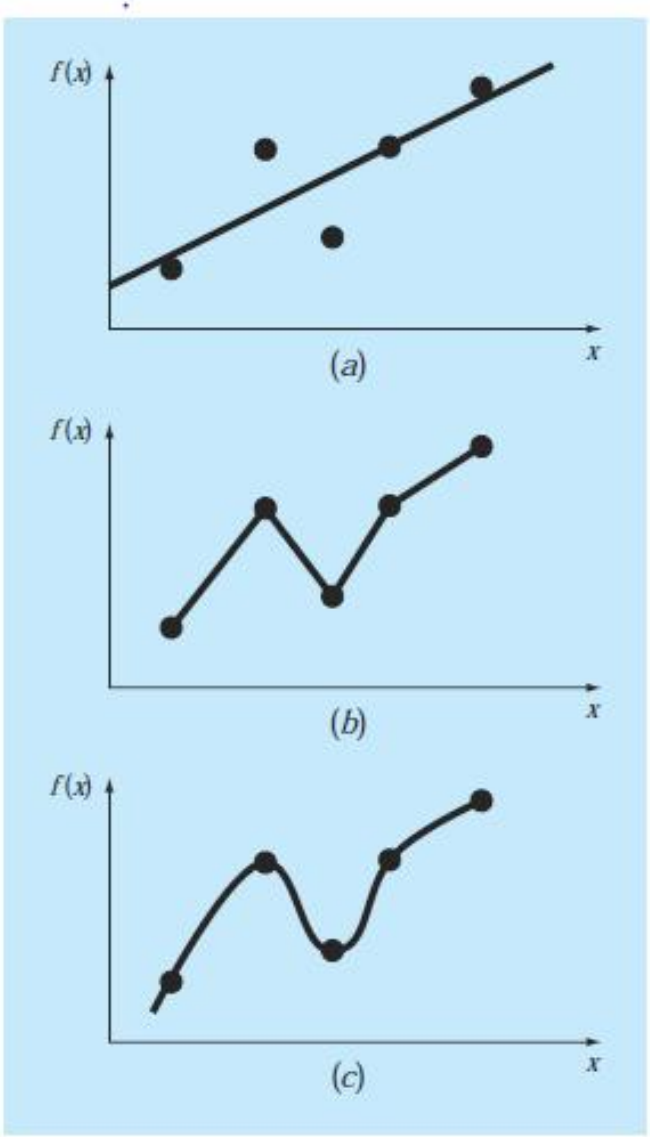
y = ax + b

(b) Linear interpolation

y1 = a1*x + b1, y2 = a2*x + b2, …

(c) Curvilinear interpolation, for example second order polynomials

a1*x**2 + b2*x + c, …



Background - Simple statistics

- Mean
- Total sum of the squared residuals
- Standard deviation
- Variance
- Coefficient of variation

$$\bar{y} = \frac{\Sigma y_i}{n}$$

$$S_t = \Sigma (y_i - \bar{y})^2$$

$$s_y = \sqrt{\frac{S_t}{n - 1}}$$

$$s_y^2 = \frac{\Sigma (y_i - \bar{y})^2}{n - 1}$$

$$\text{c.v.} = \frac{s_y}{\bar{y}} 100\%$$

Normal distribution and confidence interval(s)

Mean value = 6.6

Standard deviation = 0.1

68 % of the values between:

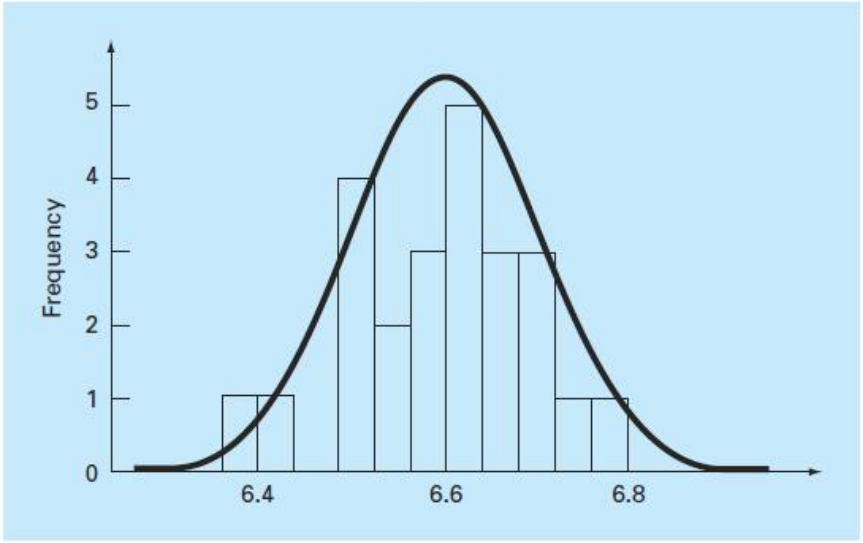
m – 1*s.d. … m + 1*s.d.

95 % of the values between:

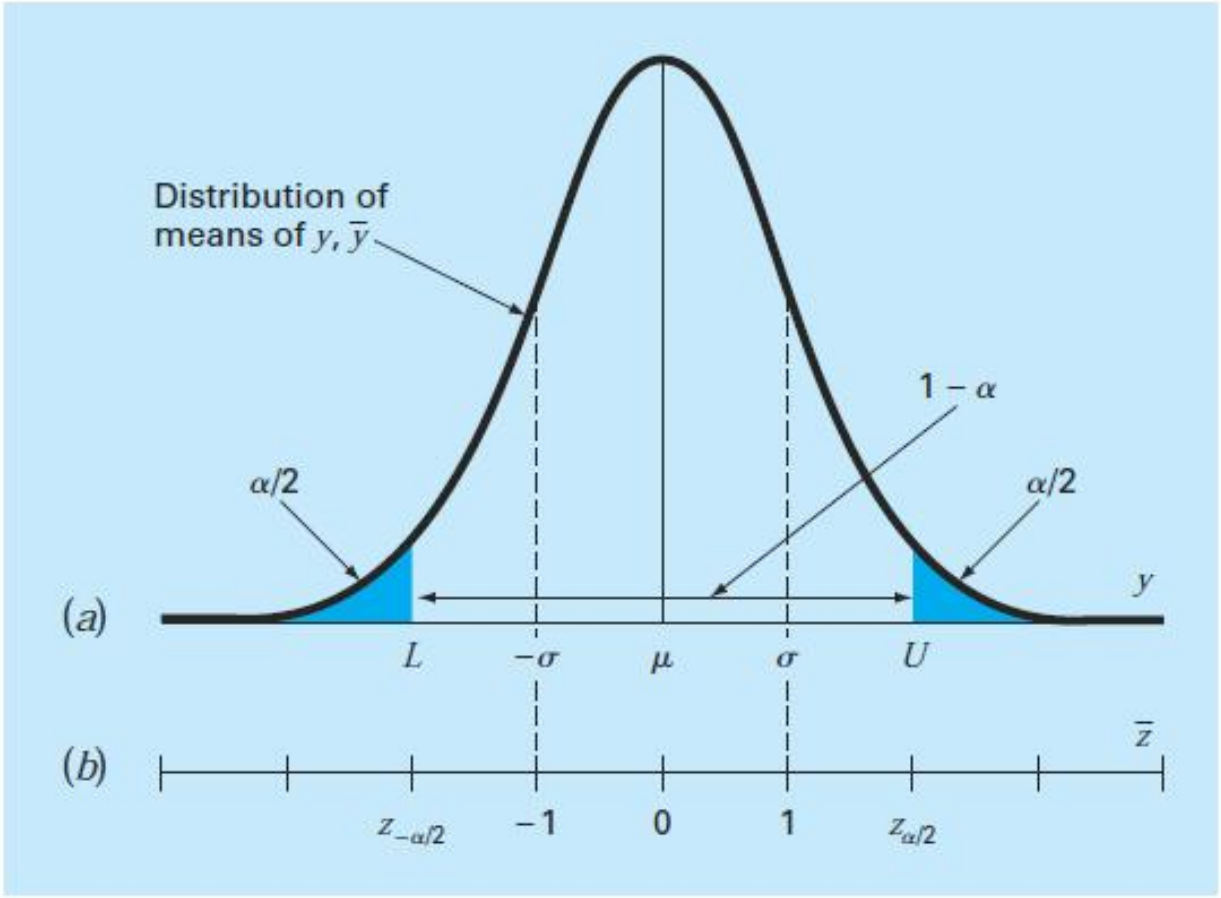
m – 1.96*s.d … m + 1.96*s.d.

99 % of the values between:

m – 3.0*s.d … m + 3.0*s.d.



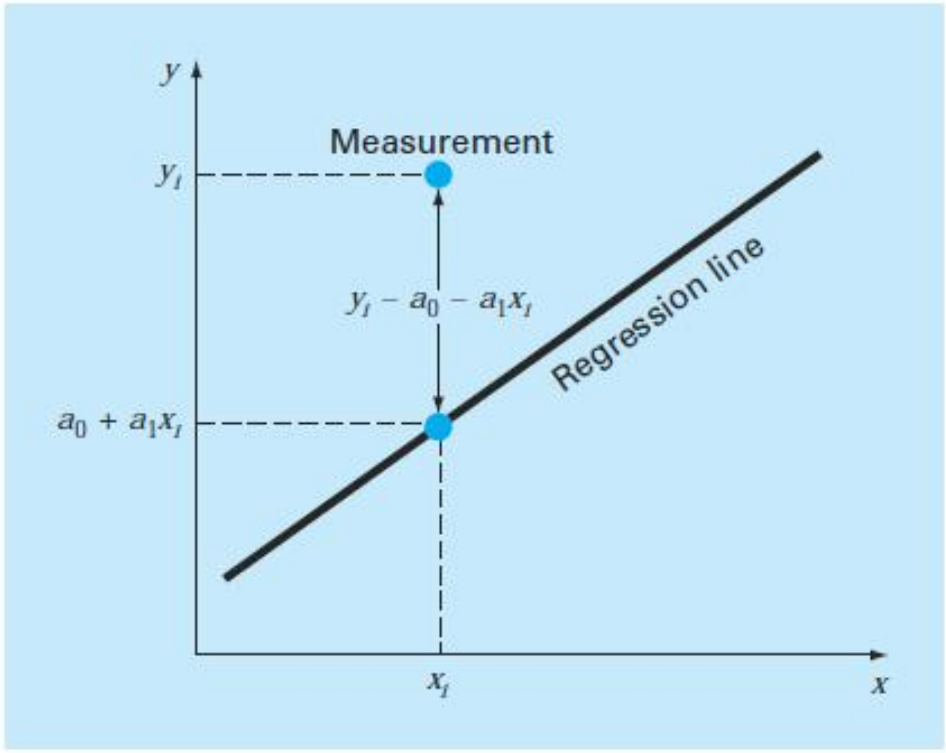
Two-sided confidence interval



Residuals and linear model

The residual represent the vertical distance between the data point and the linear model. The coefficients can be found by minimizing the total error.

$$y = a_0 + a_1 x + e$$



What is the best criteria for fitting a linear model?

Sum of the residual of the errors:

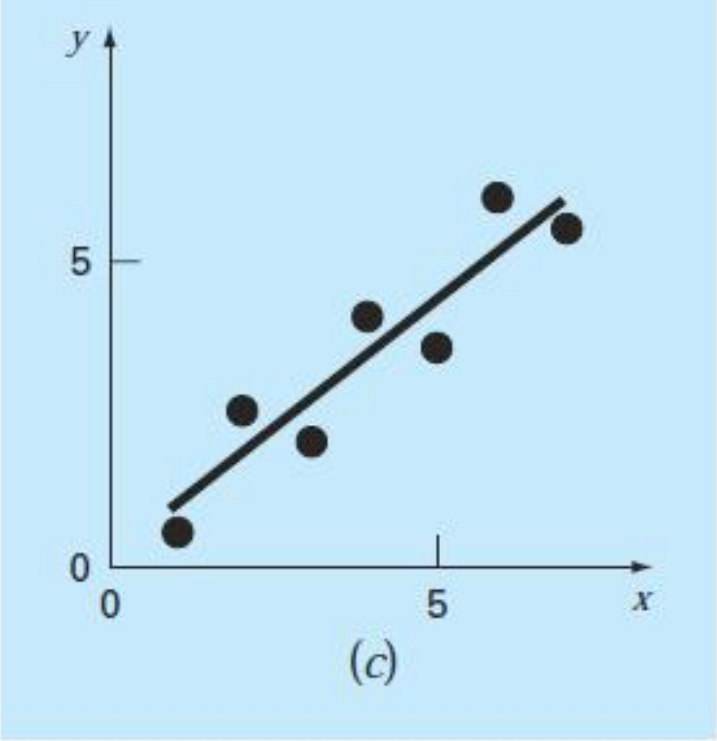
$$\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)$$

Sum of the absolute of the errors:

$$\sum_{i=1}^{n} |e_i| = \sum_{i=1}^{n} |y_i - a_0 - a_1 x_i|$$

Sum of the square of the errors:

$$\sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$



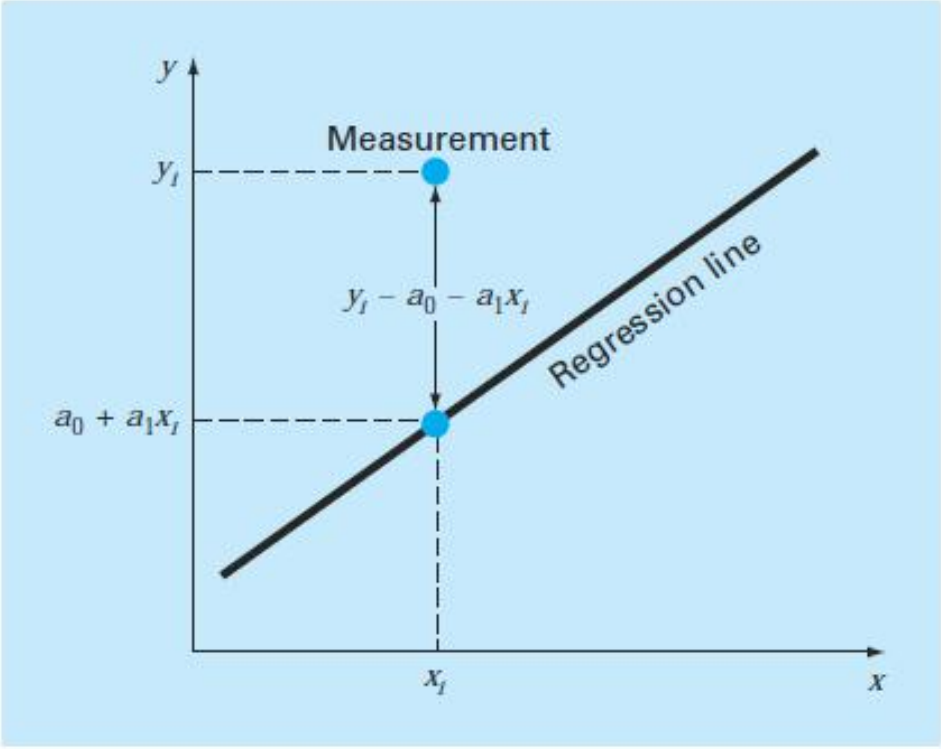
Linear reqression

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

If we use the sum of the squared residuals as the error criteria, the coefficients for the linear model can be found by minimizing the error.

$$a_1 = \frac{n\Sigma x_i y_i - \Sigma x_i \Sigma y_i}{n\Sigma x_i^2 - (\Sigma x_i)^2}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

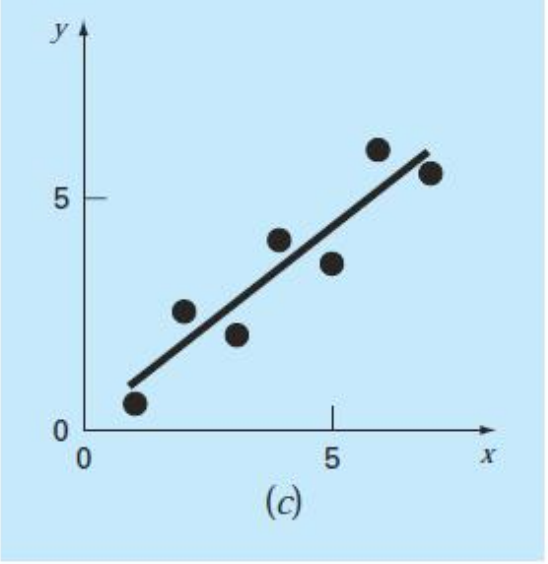


Standard error of the estimate

Standard error of the estimate e.g. the spread around the regression line can be calculated from the sum of the squared residuals.

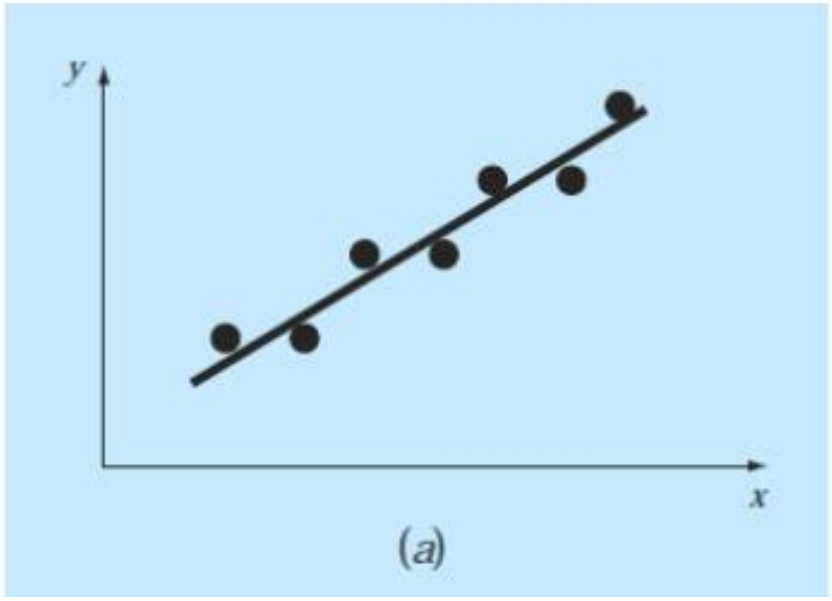
$$s_{y/x} = \sqrt{\frac{S_r}{n-2}}$$

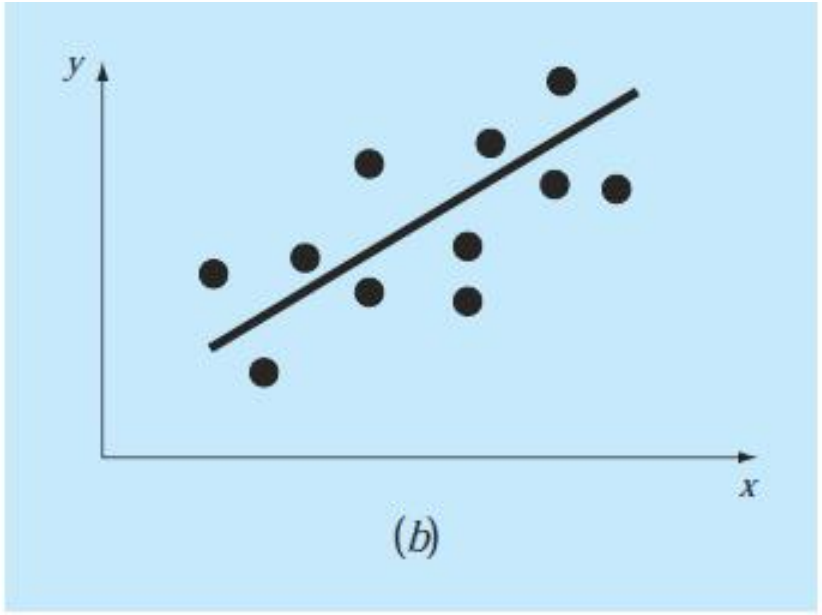
$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$



Small (a) and large (b) standard error of the estimate

$$s_{y/x} = \sqrt{\frac{S_r}{n-2}}$$





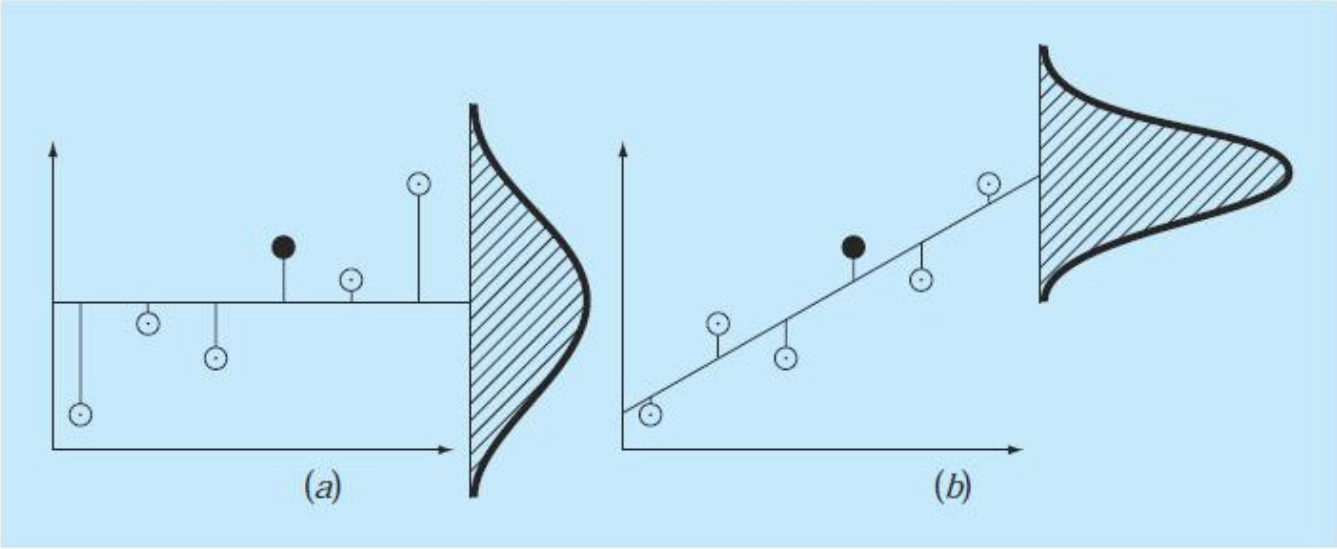
$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

Goodness of the fit

(a) Total sum of the squares around the mean $S_t = \Sigma(y_i - \bar{y})^2$

(b) Total sum of the squares of residuals $S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$

Coefficient of determination $r^2 = \frac{S_t - S_r}{S_t}$



Numpy functions

Statistics

Order statistics

amin(a[, axis, out, keepdims])	Return the minimum of an array or minimum along an axis.
amax(a[, axis, out, keepdims])	Return the maximum of an array or maximum along an axis.
nanmin(a[, axis, out, keepdims])	Return minimum of an array or minimum along an axis, ignoring any NaNs.
nanmax(a[, axis, out, keepdims])	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
ptp(a[, axis, out])	Range of values (maximum - minimum) along an axis.
percentile(a, q[, axis, out, ...])	Compute the qth percentile of the data along the specified axis.
nanpercentile(a, q[, axis, out, ...])	Compute the qth percentile of the data along the specified axis, while ignoring nan values.

Averages and variances

median(a[, axis, out, overwrite_input, keepdims])	Compute the median along the specified axis.
average(a[, axis, weights, returned])	Compute the weighted average along the specified axis.
mean(a[, axis, dtype, out, keepdims])	Compute the arithmetic mean along the specified axis.
std(a[, axis, dtype, out, ddof, keepdims])	Compute the standard deviation along the specified axis.
var(a[, axis, dtype, out, ddof, keepdims])	Compute the variance along the specified axis.
nanmedian(a[, axis, out, overwrite_input, ...])	Compute the median along the specified axis, while ignoring NaNs.
nanmean(a[, axis, dtype, out, keepdims])	Compute the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis, dtype, out, ddof, keepdims])	Compute the standard deviation along the specified axis, while ignoring NaNs.
nanvar(a[, axis, dtype, out, ddof, keepdims])	Compute the variance along the specified axis, while ignoring NaNs.

Correlating

corrcoef(x[, y, rowvar, bias, ddof])	Return Pearson product-moment correlation coefficients.
correlate(a, v[, mode])	Cross-correlation of two 1-dimensional sequences.
cov(m[, y, rowvar, bias, ddof, fweights, ...])	Estimate a covariance matrix, given data and weights.

scipy.stats.linregress

scipy.stats.linregress(*x*, *y=None*) [source]

Calculate a regression line

This computes a least-squares regression for two sets of measurements.

Parameters: **x, y** : *array_like*
two sets of measurements. Both arrays should have the same length. If only x is given (and y=None), then it must be a two-dimensional array where one dimension has length 2. The two sets of measurements are then found by splitting the array along the length-2 dimension.

Returns: **slope** : *float*
slope of the regression line
intercept : *float*
intercept of the regression line
r-value : *float*
correlation coefficient
p-value : *float*
two-sided p-value for a hypothesis test whose null hypothesis is that the slope is zero.
stderr : *float*
Standard error of the estimate

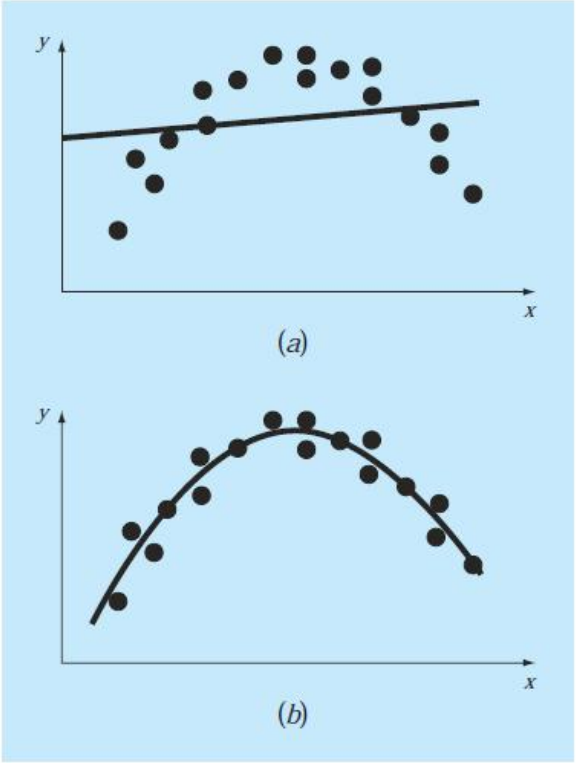
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.linregress.html

Nonlinear relationship

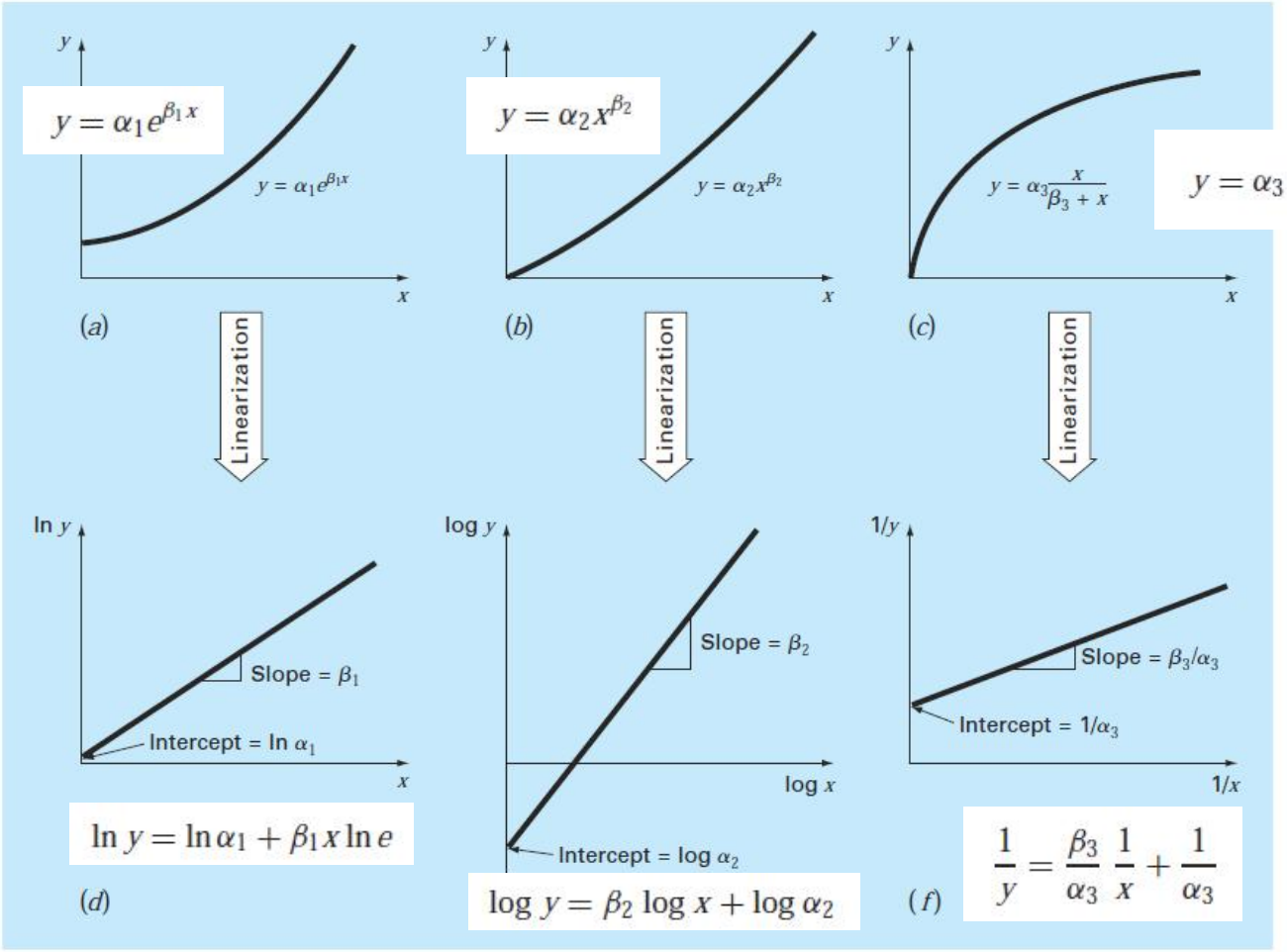
Example of data where linear (ax+b) model doesn't work, but a parabolic model (ax^2 + bx + c) would work better.

Solutions:

- Linearize the model and use linear regression
- Use nonlinear curve fitting



Linearization of nonlinear relationship



scipy.optimize.curve_fit¶

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

scipy.optimize.curve_fit(*f*, *xdata*, *ydata*, *p0=None*, *sigma=None*, *absolute_sigma=False*, *check_finite=True*, *bounds=(-inf, inf)*, *method=None*, *jac=None*, ***kwargs*) [source]

Use non-linear least squares to fit a function, f, to data.

Assumes `ydata = f(xdata, *params) + eps`

Parameters:

f : *callable*
The model function, f(x, ...). It must take the independent variable as the first argument and the parameters to fit as separate remaining arguments.

xdata : *An M-length sequence or an (k,M)-shaped array for functions with k predictors*
The independent variable where the data is measured.

ydata : *M-length sequence*
The dependent data — nominally f(xdata, ...)

p0 : *None, scalar, or N-length sequence, optional*
Initial guess for the parameters. If None, then the initial values will all be 1 (if the number of parameters for the function can be determined using introspection, otherwise a ValueError is raised).

sigma : *None or M-length sequence or MxM array, optional*
Determines the uncertainty in *ydata*. If we define residuals as `r = ydata - f(xdata, *popt)`, then the interpretation of *sigma* depends on its number of dimensions:

- A 1-d *sigma* should contain values of standard deviations of errors in *ydata*. In this case, the optimized function is `chisq = sum((r / sigma) ** 2)`.
- A 2-d *sigma* should contain the covariance matrix of errors in *ydata*. In this case, the optimized function is `chisq = r.T @ inv(sigma) @ r`.
  New in version 0.19.

None (default) is equivalent of 1-d *sigma* filled with ones.

scipy.optimize.least_squares

scipy.optimize.least_squares(*fun, x0, jac='2-point', bounds=(-inf, inf), method='trf', ftol=1e-08, xtol=1e-08, gtol=1e-08, x_scale=1.0, loss='linear', f_scale=1.0, diff_step=None, tr_solver=None, tr_options={}, jac_sparsity=None, max_nfev=None, verbose=0, args=(), kwargs={}*) [source]

Solve a nonlinear least-squares problem with bounds on the variables.

Given the residuals f(x) (an m-dimensional real function of n real variables) and the loss function rho(s) (a scalar function), least_squares finds a local minimum of the cost function F(x):

```
minimize F(x) = 0.5 * sum(rho(f_i(x)**2), i = 0, ..., m - 1)
subject to lb <= x <= ub
```

The purpose of the loss function rho(s) is to reduce the influence of outliers on the solution.

Parameters: **fun** : *callable*

Function which computes the vector of residuals, with the signature `fun(x, *args, **kwargs)`, i.e., the minimization proceeds with respect to its first argument. The argument `x` passed to this function is an ndarray of shape (n,) (never a scalar, even for n=1). It must return a 1-d array_like of shape (m,) or a scalar. If the argument `x` is complex or the function `fun` returns complex residuals, it must be wrapped in a real function of real arguments, as shown at the end of the Examples section.

x0 : *array_like with shape (n,) or float*

Initial guess on independent variables. If float, it will be treated as a 1-d array with one element.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html