Linear least squares method (*l*LSQ)

線形最小自乗法

Approximation of many sample points: Minimization (Optimization)

(多数の標本点の近似: 最小化問題)

How to determine most plausible parameters *a* and *b*
if observed data (x_1, y_1), ・・・ (x_n, y_n) follow $f(x) = a + bx$,
※ Error ε_i should be considered: $y_i = f(x_i) + \varepsilon_i$

Fundamental idea: Determine *a* and *b* so as to minimize (maximize) a target function *S* (e.g., error residual function (残差関数))

Minimax approximation — **L_∞ norm: minimize** $\max_{a \le x \le b} |f(x_i) - y_i|$

Minimize — **L_1 norm : $S = \Sigma|f(x_i) - y_i|$**

Least-squares (LSQ) method (最小自乗法) — **L_2 norm : $S = \Sigma(f(x_i) - y_i)^2$**

L_n norm:
$S_n = \sum_i |f(x_i) - y_i|^n$
L_0 norm: $S_0 = 0$

Linear least-squares method (線形最小二乗法/線形回帰)

Linear problem:

Model function $f(x_i) = a + bx_i$ is a linear function of fitting parameters a and b:

Not necessary to be a linear function of x_i

Minimize the objective function (loss function) $S = \Sigma(a + bx_i - y_i)^2$

$$dS/da = 2\Sigma(a + bx_i - y_i) \quad = 2an \quad + 2b\Sigma x_i - 2\Sigma y_i \quad = 0$$

$$dS/db = 2\Sigma x_i(a + bx_i - y_i) = 2a\Sigma x_i + 2b\Sigma x_i^2 - 2\Sigma x_i y_i = 0$$

※ **Solve simultaneous equation**

$$2n \cdot a \quad + 2\Sigma x_i \cdot b \quad = 2\Sigma y_i$$

$$2\Sigma x_i \cdot a + 2\Sigma x_i^2 \cdot b = 2\Sigma x_i y_i$$

Matrix representation would be easy to understand:

$$\begin{pmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}$$

(a, b) are obtained by solving the 2x2 matrix equation

Even for $f(x) = a + bx + cx^2 + \cdots$, only one matrix operation can give a final solution

*l*LSQ: Polynomial

線形最小二乗法: 多項式

$$f(x) = \sum_{k=0}^{n} a_k x^k \qquad S = \sum_{i=1}^{N} \left(y_i - \sum_{k=0}^{n} a_k {x_i}^k \right)^2$$

$$\frac{dS}{da_l} = -2 \sum_{i=1}^{N} {x_i}^l \left(y_i - \sum_{k=0}^{n} a_k {x_i}^k \right) = 0$$

$$\sum_{k=0}^{n} \sum_{i=1}^{N} a_k {x_i}^{k+l} = \sum_{i=1}^{N} y_i {x_i}^l \qquad (l = 0, 1, \cdots, N)$$

$$\begin{pmatrix} n & \sum x_i & \sum {x_i}^2 & \cdots & \sum {x_i}^N \\ \sum x_i & \sum {x_i}^2 & \sum {x_i}^3 & & \sum {x_i}^{N+1} \\ \sum {x_i}^2 & \sum {x_i}^3 & \sum {x_i}^4 & & \sum {x_i}^{N+2} \\ \vdots & & & \ddots & \\ \sum {x_i}^N & \sum {x_i}^{N+1} & \sum {x_i}^{N+2} & & \sum {x_i}^{2N} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i {x_i}^2 \\ \vdots \\ \sum y_i {x_i}^N \end{pmatrix}$$

$|x_i| > 1$ might cause overflow,
$|x_i| < 1$ might cause underflow errors.

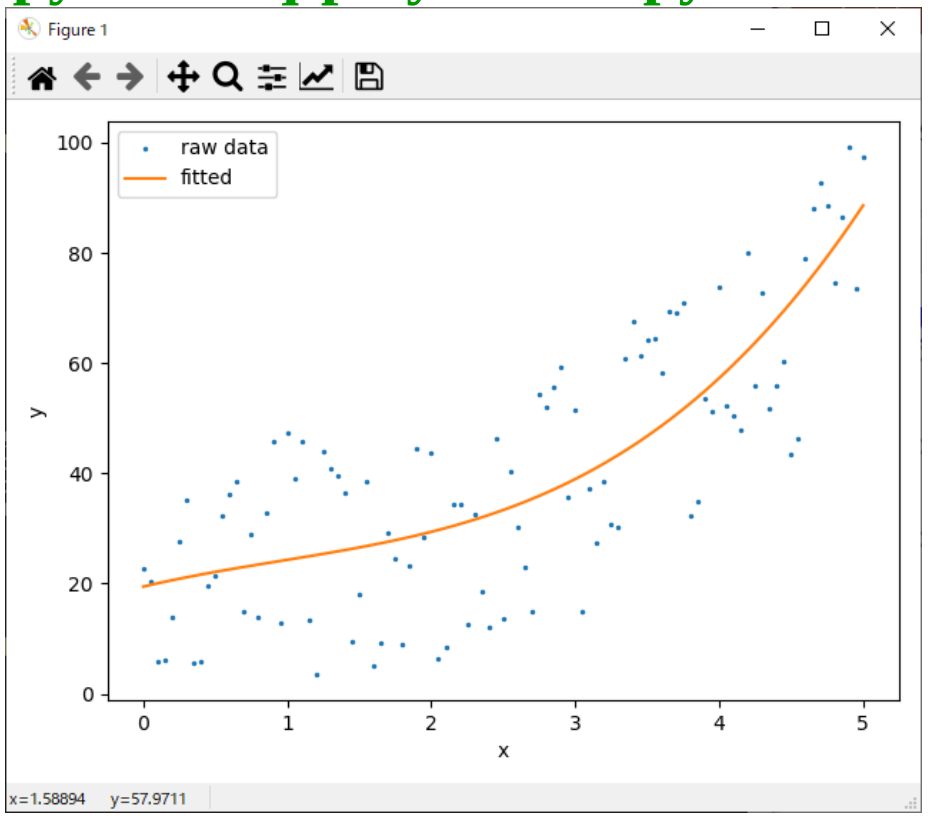
=> Normalize the x range e.g. to [-1, 1] : $x_i' = 2 \frac{x_i - x_{\mathrm{mid}}}{x_{\max} - x_{\min}}$

by average and standard deviation: ${x_i}' = 2 \frac{x_i - x_{\mathrm{average}}}{\sigma_x}$

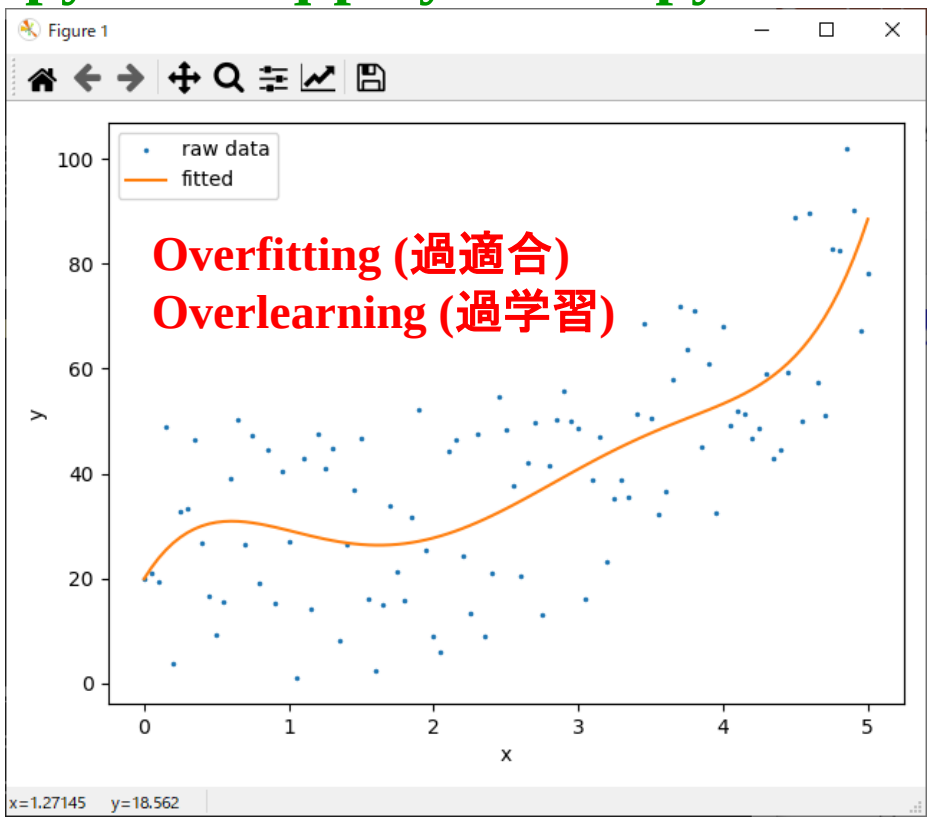
Program: lsq-polynomial.py

Usage: python lsq-polynomial.py n_{order}

python lsq-polynomial.py 3



python lsq-polynomial.py 6



*l*LSQ: General functions (任意の関数)

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_p f_p(x) = \sum_{k=0}^{p} a_k f_k(x)$$

$f(x)$ is a linear function w.r.t. a_i, applicable to arbitrary base functions $f_k(x)$.

e.g., $f(x) = a + b\log x + c/x, f(x,y) = a + bxy + cy/x$

Minimize $S = \sum_{j=1}^{n} (\sum_{k=1}^{p} a_k f_k(x_j) - y_j)^2$

$$\begin{pmatrix} \sum f_1(x_i)f_1(x_i) & \sum f_1(x_i)f_2(x_i) & \sum f_1(x_i)f_3(x_i) & \cdots & \sum f_1(x_i)f_p(x_i) \\ \sum f_2(x_i)f_1(x_i) & \sum f_2(x_i)f_2(x_i) & \sum f_2(x_i)f_3(x_i) & & \sum f_2(x_i)f_p(x_i) \\ \sum f_3(x_i)f_1(x_i) & \sum f_3(x_i)f_2(x_i) & \sum f_3(x_i)f_3(x_i) & & \sum f_3(x_i)f_p(x_i) \\ \vdots & & & \ddots & \\ \sum f_p(x_i)f_1(x_i) & \sum f_p(x_i)f_2(x_i) & \sum f_N(x_i)f_3(x_i) & & \sum f_p(x_i)f_p(x_i) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i f_1(x_i) \\ \sum y_i f_2(x_i) \\ \sum y_i f_3(x_i) \\ \vdots \\ \sum y_i f_p(x_i) \end{pmatrix}$$

$$D_{k,k'} = \sum_{j=1}^{n} f_{k'}(x_j) f_k(x_j) \qquad A_k = a_k \quad V_k = \sum_{j=1}^{n} y_j f_k(x_j)$$

Solve $DA = V$

$X = (f_j(x_i))$: Design Matrix

$$D = X^t X, V = X^t Y, Y_{predict} = X^t A$$

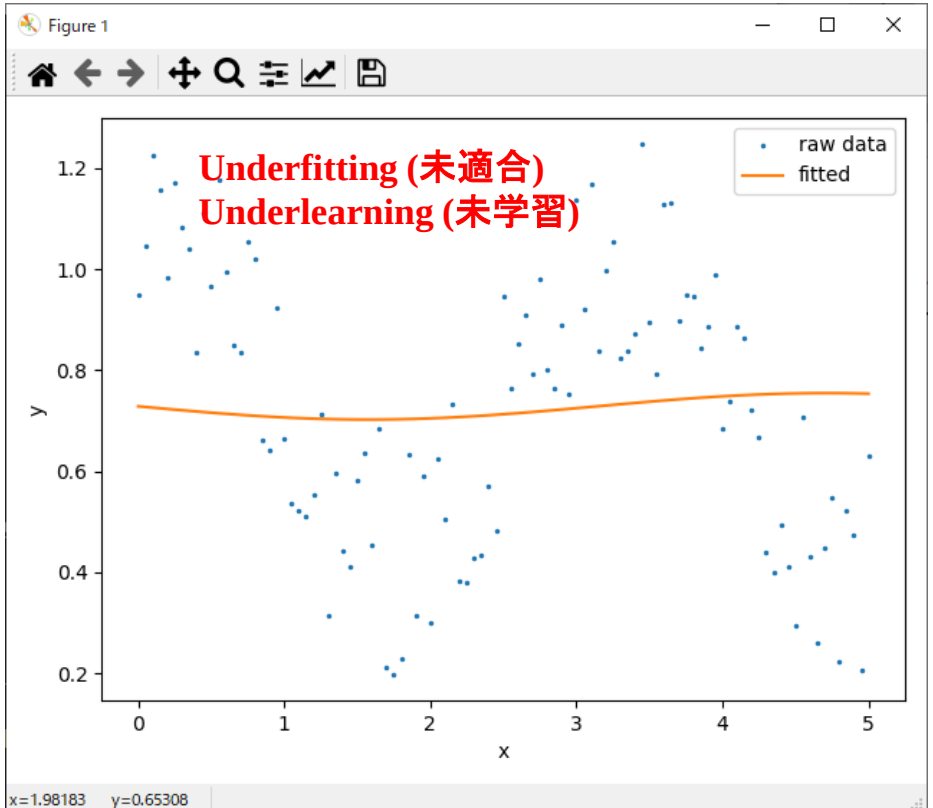
Program: lsq-general.py

Usage: python lsq-general.py n_{func}

fit to $y = c_0 + c_1 \sin x + c_2 \cos x + c_3 \sin 2x + c_4 \cos 2x + c_5 \sin 3x + c_6 \cos 3x$

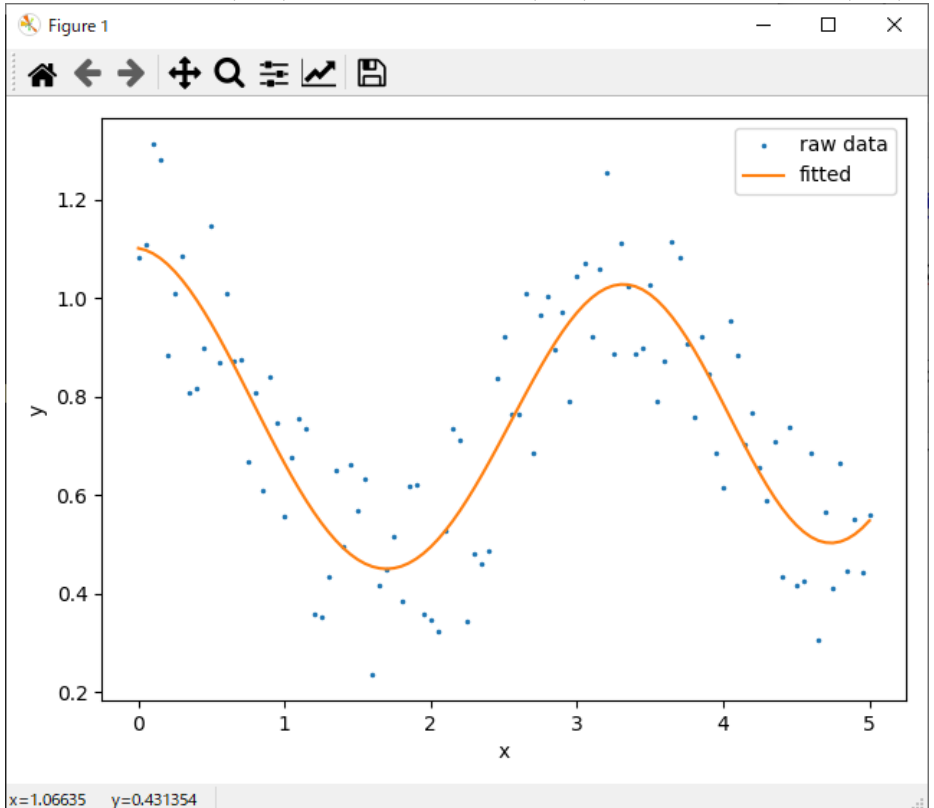
python lsq-general.py 2

$y = 0.740 + 0.000432 \sin(x)$



python lsq-polynomial.py 6

$y = 0.753 + 0.0064 \sin(x) + 0.00358 \cos(x) + 0.125 \sin(2x) + 0.303\cos(2x) + 0.0119\sin(3x)$



Ex of *l*LSQ: Lattice spacing of triclinic lattice

(三斜晶結晶の面間隔)

$$d_{hkl}^{-2} = |\mathbf{G}_{hkl}|^2 = |h\mathbf{a}^* + k\mathbf{b}^* + l\mathbf{c}^*|^2$$

$$\frac{1}{d_{hkl}^{\ 2}} = S_{11}h^2 + S_{22}k^2 + S_{33}l^2 + 2S_{12}hk + 2S_{23}kl + 2S_{31}lh$$

$$S_{11} = \mathbf{a}^* \cdot \mathbf{a}^* = b^2c^2\sin^2\alpha / V^2$$

$$S_{22} = c^2a^2\sin^2\beta / V^2$$

$$S_{33} = c^2a^2\sin^2\gamma / V^2$$

$$S_{12} = \mathbf{a}^* \cdot \mathbf{b}^* = abc^2\left(\cos\alpha\cos\beta - \cos\gamma\right)/V^2$$

$$S_{23} = a^2bc\left(\cos\beta\cos\gamma - \cos\alpha\right)/V^2$$

$$S_{31} = ab^2c\left(\cos\gamma\cos\alpha - \cos\beta\right)/V^2$$

$$V = abc\sqrt{1 - \cos^2\alpha - \cos^2\beta - \cos^2\gamma + 2\cos\alpha\cos\beta\cos\gamma}$$

The form of d_{hkl}^{-2} is a linear function with respect to S_{ij}.

1. S_{ij} is obtained by *l*LSQ
2. S_{ij} => Reciprocal lattice parameters (a^*, b^*, c^*, α^*, β^*, γ^*)
3. => Lattice parameters (a, b, c, α, β, γ)

Handling Outliers (外れ値の取り扱い)

Least-squares (LSQ) method: minimize $S = \Sigma(f(x_i) - y_i)^2 = \Sigma r_i^2$

- Large residuals make a disproportionately large contribution because they are squared.
- The largest residual is not explicitly bounded.

Minimax approximation: minimize $\max_i |f(x_i; a) - y_i|$

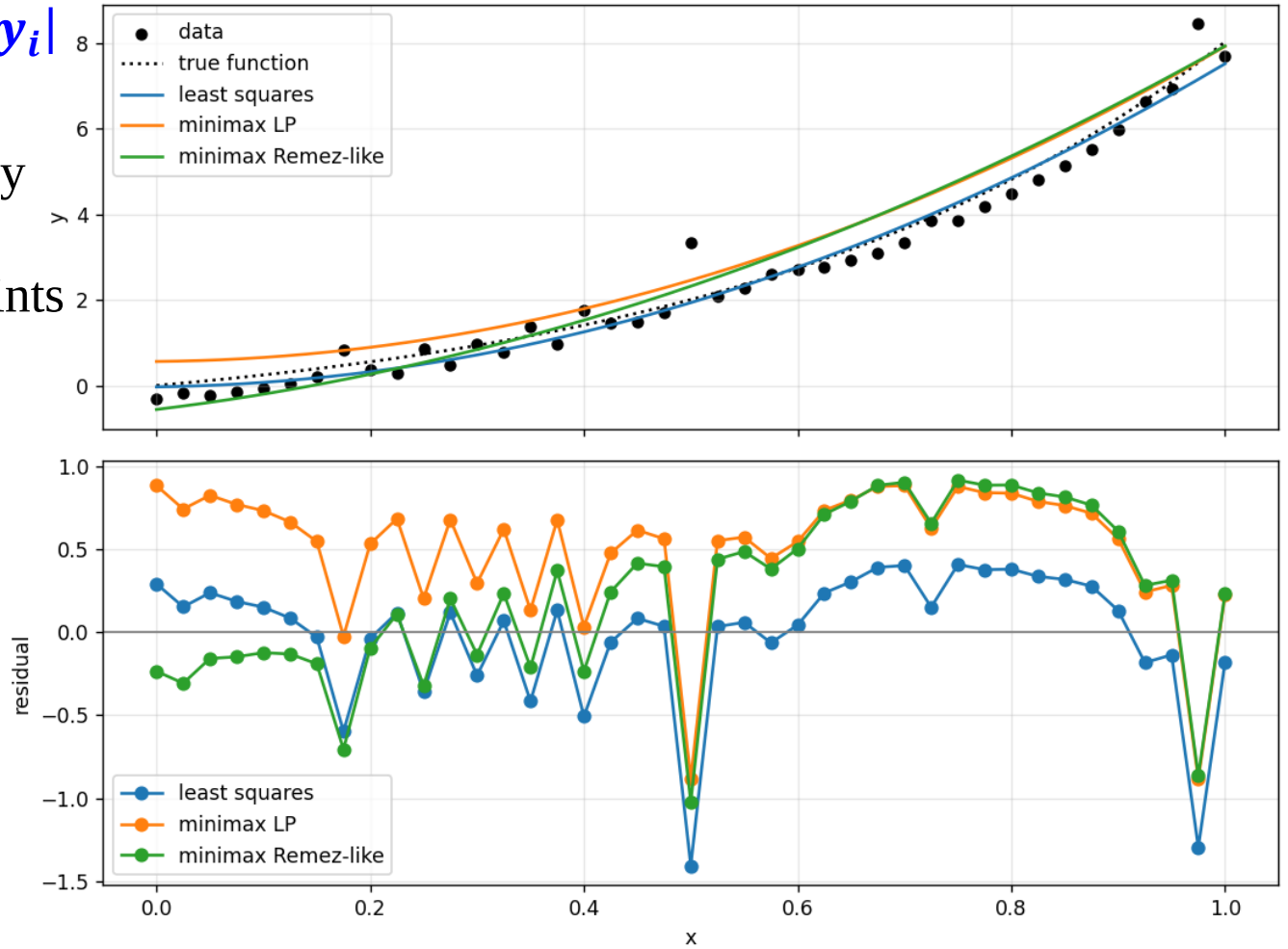
- Guarantees the largest deviations are minimum
- Outliers change the regression curve very significantly

Algorisms:

(A) Linear programming (LP) with inequality constraints

(B) Remetz(-like) method

python minimax.py --tnoise z2 --knoise .5



Robust Regression in the Presence of Outliers (外れ値に対するロバスト回帰)

Huber regression (Huber回帰) using IRLS (Iteratively Reweighted Least Squares)

1. Perform a least-squares fit with the initial weights ($w_i = 1$)
2. Calculate residuals r_i
3. Downweight data points with large residuals.
   (残差が大きい点の重みを小さくする)
   Example for Huber regression:

$$w_i = \begin{cases} 1 & if \ |r_i| \le \delta, \\ \delta / |r_i| & otherwise. \end{cases}$$

4. Solve the weighted least-squares problem again using the updated weights.
   (新しい重みで再び最小二乗法を解く)
   Return to Step 2 and repeat until convergence.

huber_irls.py