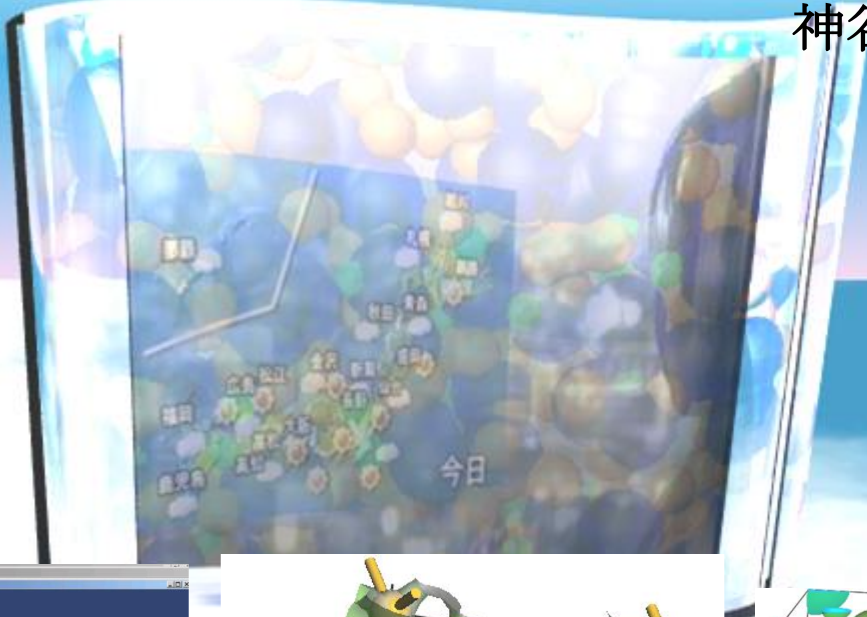
Computational Materials Science (計算材料学特論)

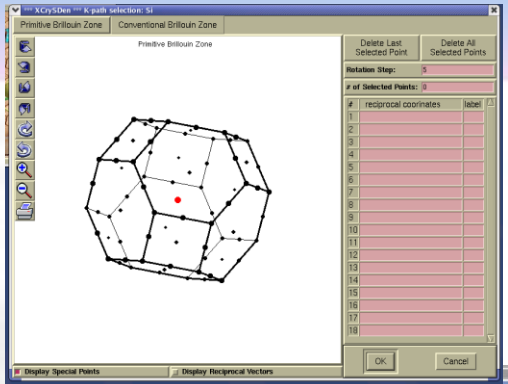
No update today: Lecture materials

Will start ~8:52

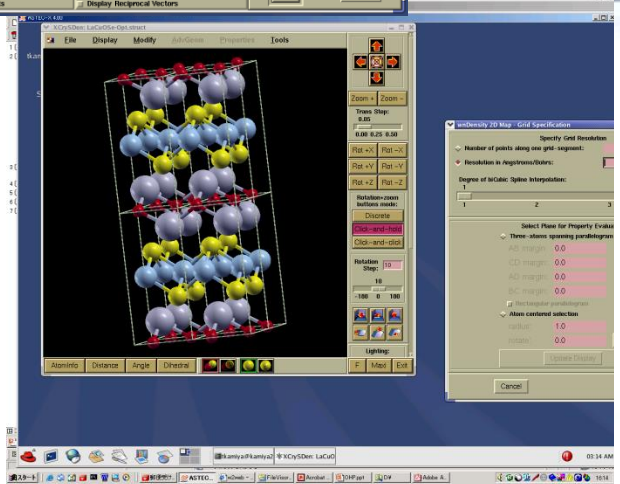
Computational Materials Science
計算材料学特論

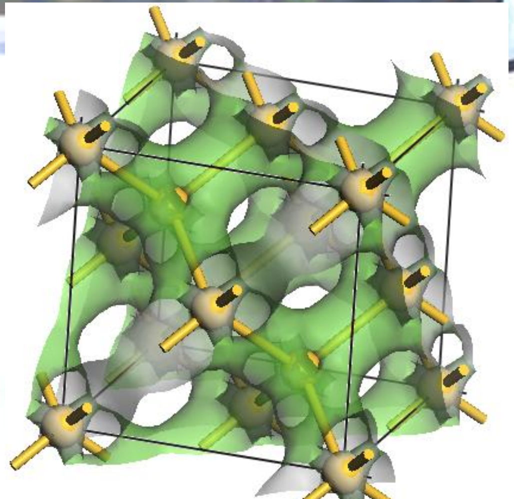
Toshio Kamiya
神谷利夫

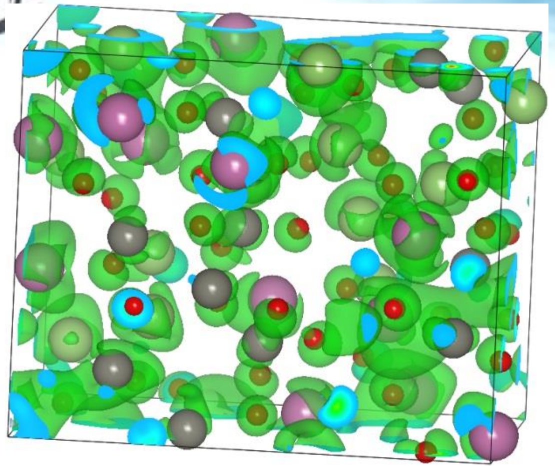



XCrySDen K-path selection: Si
Primitive Brillouin Zone
Conventional Brillouin Zone
Delete Last Selected Point
Delete All Selected Points
reciprocal coorinates
label
OK
Cancel
Display Special Points
Display Reciprocal Vectors

Rotation Step: 5
# of Selected Points: 0







Class Schedule

Lecture materials (Kamiya's part): http://conf.msl.titech.ac.jp/Lecture/

http://conf.msl.titech.ac.jp/Lecture/ComputationalMaterialsScience/index-numericalanalysis.html

#01 June 11 (Tue)	Kamiya (Fundamental of computer, Sources of errors (コンピュータの基礎、誤差))
#02 June 14 (Fri)	Kamiya (Numerical differentiation/integration (数値微分/積分))
#03 June 18 (Tue)	Kamiya (Numerical integration (数値積分), Differential equation (微分方程式), Molecular dynamics (分子動力学法))
#04 June 21 (Fri)	Kamiya (Interpolation (補間), Smoothing (平滑化), Linear least-squares method (線形最小二乗法))
#05 June 25 (Tue)	Kamiya (solutions of equations (方程式の解), Optimization (最適化), Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化))
#06 June 28 (Fri)	Kamiya (Fourier transformation (フーリエ変換) , Matrix (行列))
July 2 (Tue)	**No lecture (休講)**
#07 July 5 (Fri)	Kamiya, Review (復習)
#08 July 9 (Tue)	Sasagawa (Review of quantum theory 1: 量子論おさらい1)
#09 July 12 (Fri)	Sasagawa (Review of quantum theory 2: 量子論おさらい2)
#10 July 16 (Tue)	Sasagawa (First principles calculations: basics 1 第一原理計算:基礎1)
#11 July 19 (Fri)	Sasagawa (First principles calculations: basics 2 第一原理計算:基礎2)
#12 July 23 (Tue)	Sasagawa (First principles calc.: applications 1 第一原理計算:応用1)
#13 July 26 (Tue)	Sasagawa (First principles calc.: applications 2 第一原理計算:応用2)
#14	Sasagawa (Classical and Quantum Computers 古典および量子コンピュータ)

Explanation of the answers, June 18

課題解答の解説

PROBLEM, June 18

PROBLEM:

(i) By filling the *dx*/*dt* and the *x*(*t*) columns in diffeq.xlsx, solve *dx*(*t*) / *dt* = –*x*(*t*)sin(πt) using the Euler method.

Conditions:

t starts from 0 and ends at 3.0 with the time step of 0.1.

$x(0) = 1.0$

Euler formula: $\frac{dx(t)}{dt} = f(x(t), t) = -x(t)\sin(\pi t)$

$$x(t+\Delta t) = x(t) + \Delta t \cdot f(\boldsymbol{x(t)}, \boldsymbol{t})$$

Typical mistake:

$$x(t+\Delta t) = x(t) + \Delta t \cdot f(x(\boldsymbol{t+\Delta t}), \boldsymbol{t+\Delta t})$$

This calculation is not possible if $f(x, t)$ includes x explicitly.

See diffeq2_answer.xlsx

PROBLEM, June 21

- Submit electronic file(s) via T2SCHOLAR in 2 days
  (If T2SCHOLAR doesn’t work, send the files to kamiya.t.aa@m.titech.ac.jp.
  In this case, file name must include your STUDENT ID and FULL NAME)

PROBLEM:

Smoothen the data DOS(E) in dos.xlsx
by simple average method and polynomial fit method.

Add them and plot the raw DOS(E) and the smoothed data in an Excel file.

You can choose smoothing parameters as you like, but explicitly descrive them.

Submit the excel file.

Approximation of discrete data: Interpolation/Extrapolation

(離散データの近似: 補間/補外）

Interpolation

Pattern 1: Reproduce all sample points (標本点を必ず通る)

n sample points are reproduced by (*n* – 1) order polynomial.

・Interpolated data might be scattered largely in particular for orders higher than 3 (Runge's phenomenon/oscillation **ルンゲの現象**).

補間点が大きく振動する問題がでる。特に3次以上の多項式

=> To suppress the Runge's phenomenon:

Make the *n*-th order differentiations continuous at the boundaries between neighboring regions

=> Spline function

n sample points are reproduced by (*n* + *N* – 1) order polynomial.

Pattern 2: Smoothing (平滑化)

Scattering of data will be reduced

Pattern 3: Does not reproduce sample points exactly, but the deviation will be minimized

(標本点を通らないが、補間データは標本点から大きく外れない)

・Least-squares method (LSQ, 最小二乗法)

・Minimax approximation (ミニマックス近似)

Polynomial that reproduces sample points
(標本点を通る多項式)

n sample points (x_i, y_i) $(i = 1, \cdots, n)$ are reproduced by $(n - 1)$ order polynomial.

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

$$\begin{pmatrix} 1 & x_1 & {x_1}^2 & \cdots & {x_1}^{n-1} \\ 1 & x_2 & {x_2}^2 & & {x_2}^{n-1} \\ 1 & x_3 & {x_3}^2 & & {x_3}^{n-1} \\ \vdots & & & \ddots & \\ 1 & x_n & {x_n}^2 & & {x_n}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$|x_i| > 1$ might cause overflow,
$|x_i| < 1$ might cause underflow errors.
=> Normalize (正規化) the *x* range e.g. to [-1, 1] : $x_i' = 2\frac{x_i - x_{\mathrm{mid}}}{x_{\mathrm{max}} - x_{\mathrm{min}}}$

Standardize (標準化) by average and standard deviation: ${x_i}' = 2\frac{x_i - x_{\mathrm{average}}}{\sigma_x}$

Lagrange interpolation formula

(ラグランジの補間公式)

戸田英雄, 小野令美，入門 数値計算, オーム社 (昭和58年)

(n – 1) order polynomial that reproduces n sample points (x_i, y_i) (i = 0, ･･･, n – 1) is determined uniquely.

Lagrange interpolation formula

$$P_{n-1}(x) = f(x_0)\phi_0(x) + f(x_1)\phi_1(x) + \cdots f(x_{n-1})\phi_{n-1}(x)$$

$$\phi_i(x) = \frac{\prod_{k\neq i}^{n-1}(x-x_k)}{\prod_{k\neq i}^{n-1}(x_i-x_k)} = \prod_{k\neq i}^{n-1}\frac{(x-x_k)}{(x_i-x_k)}$$

n = 2:

$$P_1(x) = f(x_0)\frac{(x-x_1)}{(x_0-x_1)} + f(x_1)\frac{(x-x_0)}{(x_1-x_0)}$$

n = 3:

$$P_2(x) = f(x_0)\frac{(x-x_1)}{(x_0-x_1)}\frac{(x-x_2)}{(x_0-x_2)} + f(x_1)\frac{(x-x_0)}{(x_1-x_0)}\frac{(x-x_2)}{(x_1-x_2)} + f(x_2)\frac{(x-x_0)}{(x_2-x_0)}\frac{(x-x_1)}{(x_2-x_1)}$$

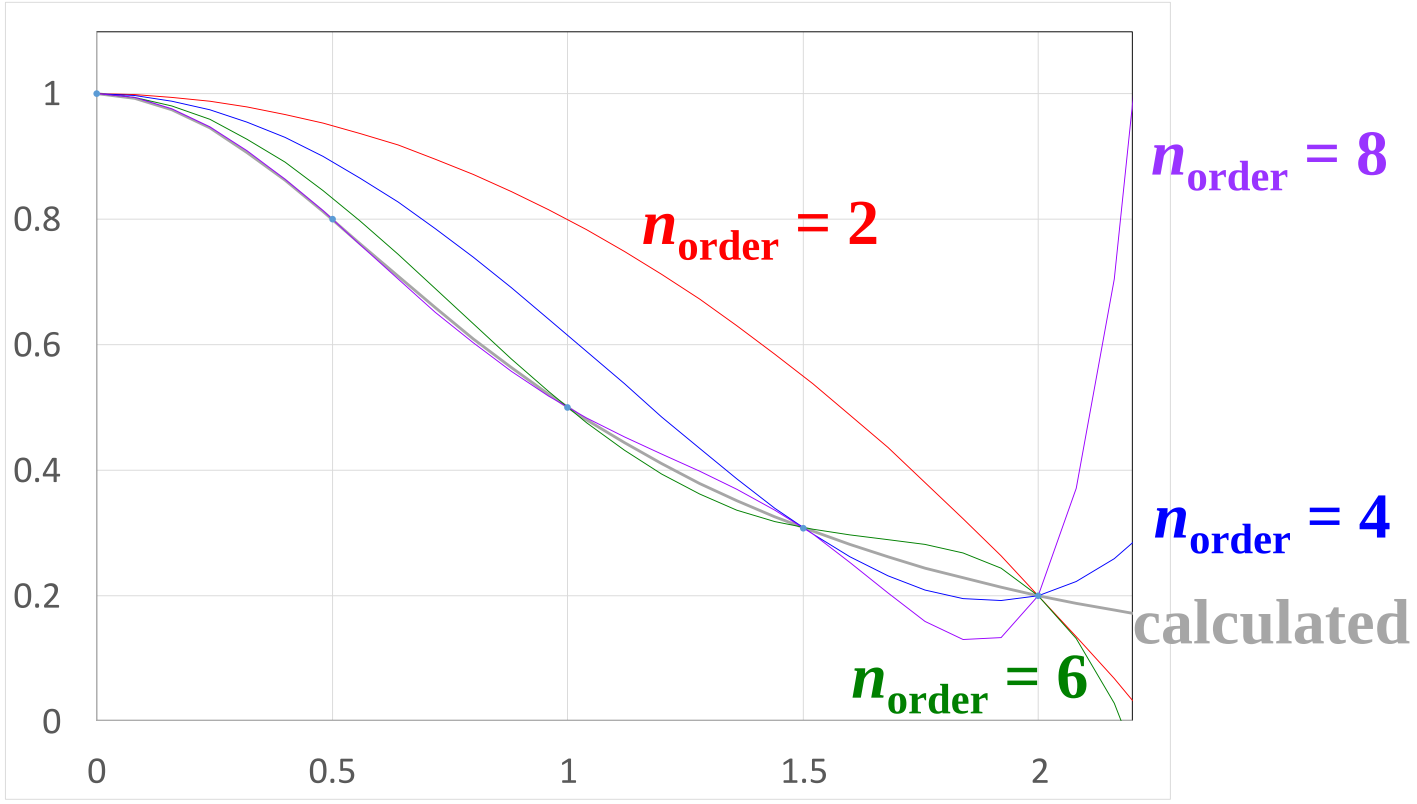
Problem of such polynomials

・Increasing the sample points will change the coefficients of polynomial completely.

・Runge's phenomenon / oscillation (ルンゲの現象)

High order (e.g. >3) polynomial will cause large oscillations at points other than the sample points (高次の多項式では標本点以外で大きく振動することがある)

***Ex.* Interpolate $f(x) = 1 / (1 + x^2)$ for (n_{order}+1) points in the range x = [-2, 2]**



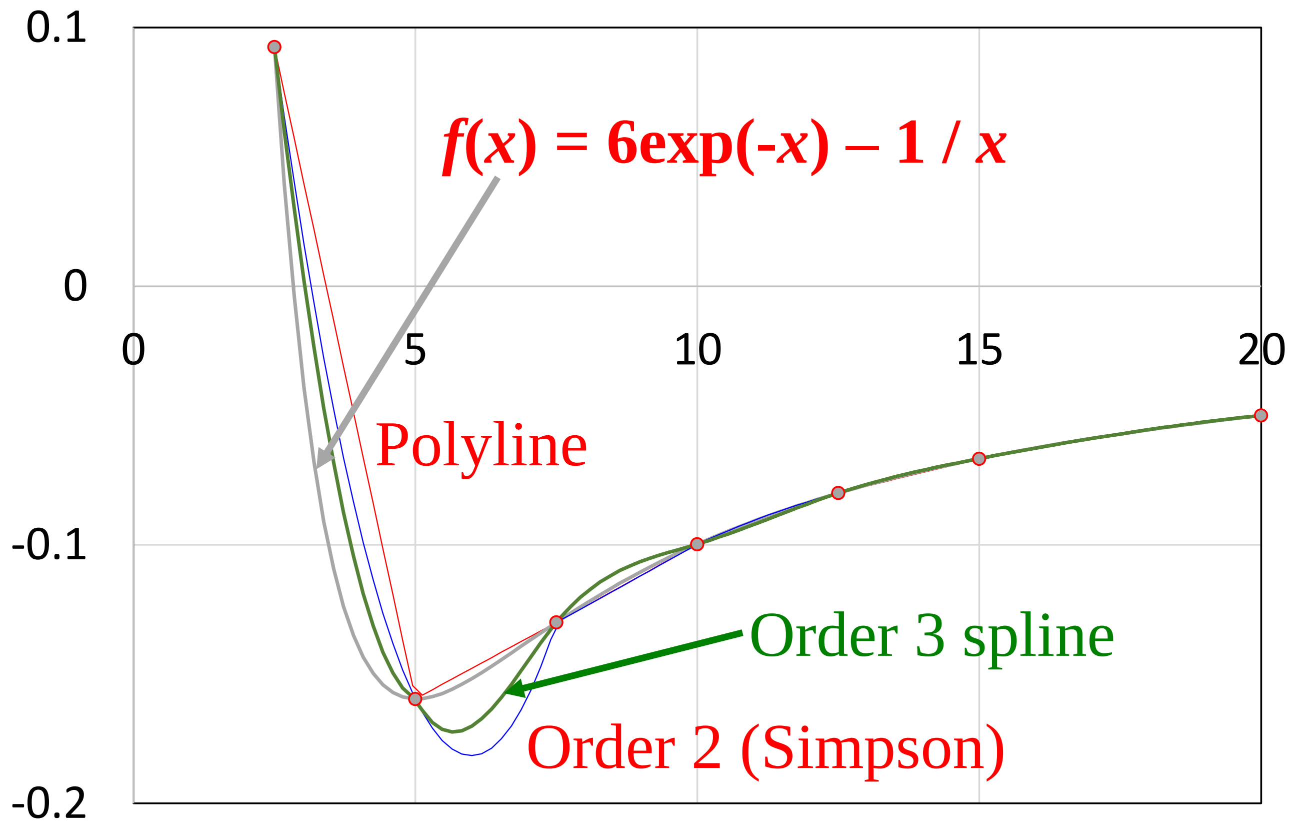
In the machine learning (機械学習):

Overfitting (過適合), Overlearning (過学習)

Interpolation: Piecewise polynomial interpolation
(区分多項式補間)

Connect divided sections by polylines (折れ線)
=> First derivatives will be discontinuous at the boundaries

=> (n – 1)-th derivatives are continuous for whole range:
Order *n* spline functions (n次のスプライン関数)



Smoothing

平滑化

Smoothing (平滑化)

Take some average for sample points

- Moving average (移動平均)
  - Simple moving average (単純移動平均):
    Average of sequential data with the uniform weight
  - Weighted moving average (加重移動平均):
    Average of sequential data with weight
    Weight：Linear, Triangular, Exponetnial, Gauss, etc…

Approximate sample points by some function

- Polynomial smoothing (多項式による平滑化)
- Smoothing spline (スプライン平滑化)
- Least-squares method (最小二乗法)

Other

- Fourier transformation (フーリエ変換)

Calculation

Simple moving average (2m+1 points)

$$y_{i,smoothed} = \frac{1}{2m+1} \sum_{j=i-m}^{i+m} y_j$$

Weighted moving average (2m+1 points)

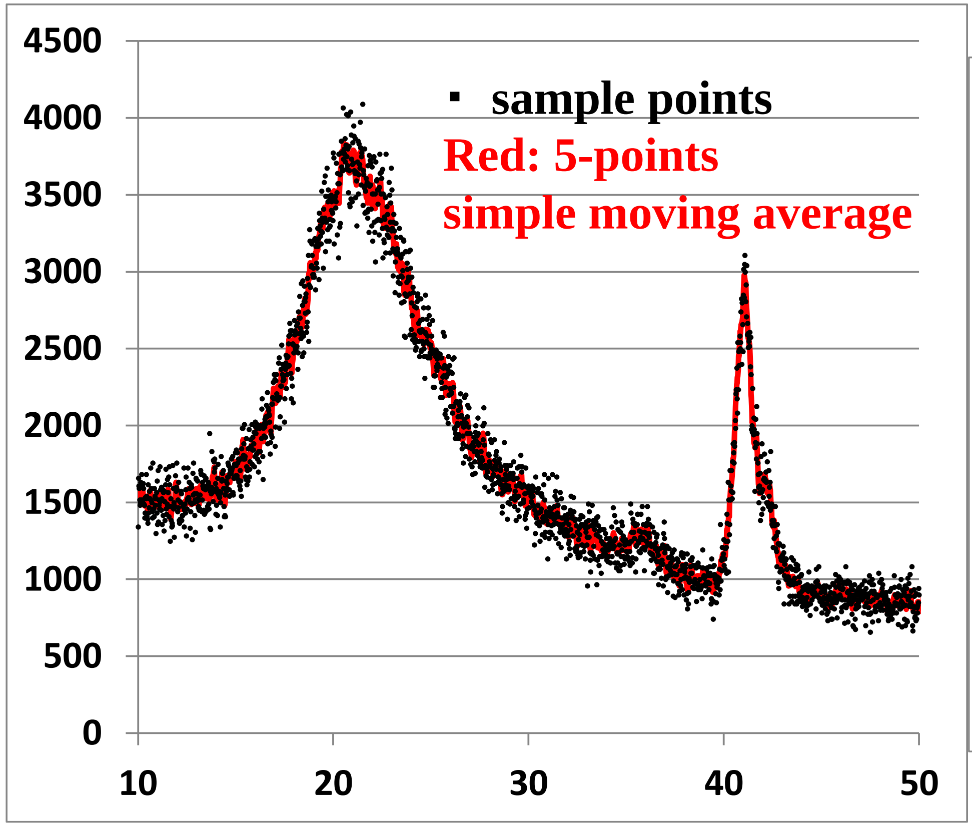
$$y_{i,smoothed} = \sum_{j=i-m}^{i+m} w_j y_j \,/\, \sum_{j=i-m}^{i+m} w_i$$

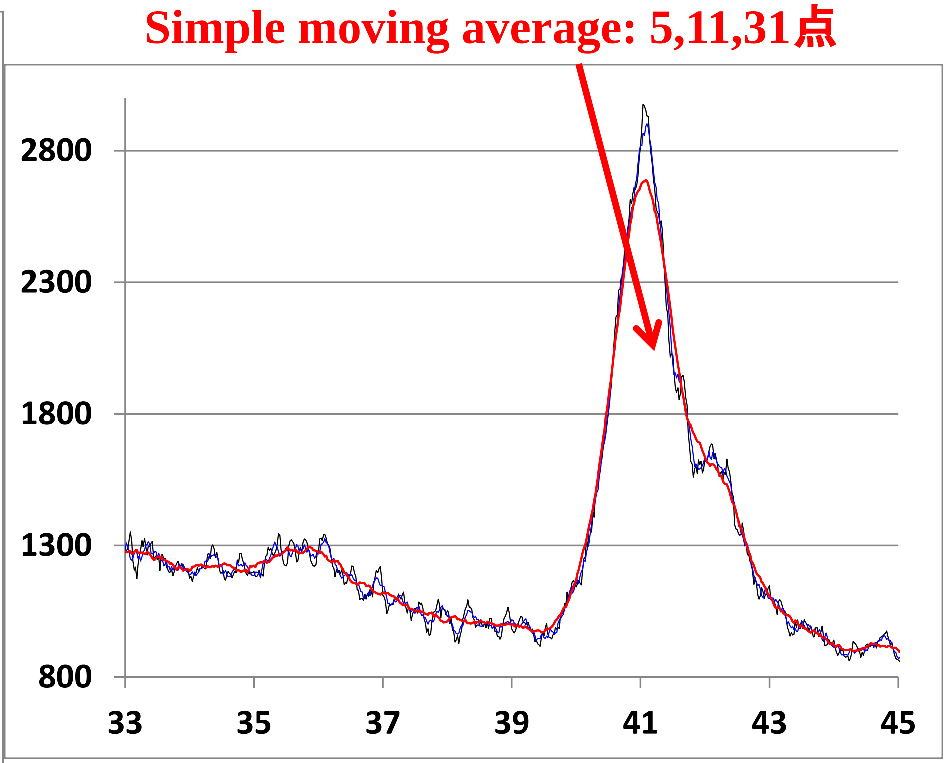
Smoothing (平滑化)

- **Moving average** (移動平均法)

  Smoother with more sample points for average,
  but would alter the function shape if the function is not monotonic.
  => Affect peak height, valley depth, peak width etc…
  The range of averaged sample points larger than the peak width
  => split peaks might become difficult to be separated.

Poor S/N ratio XRD pattern



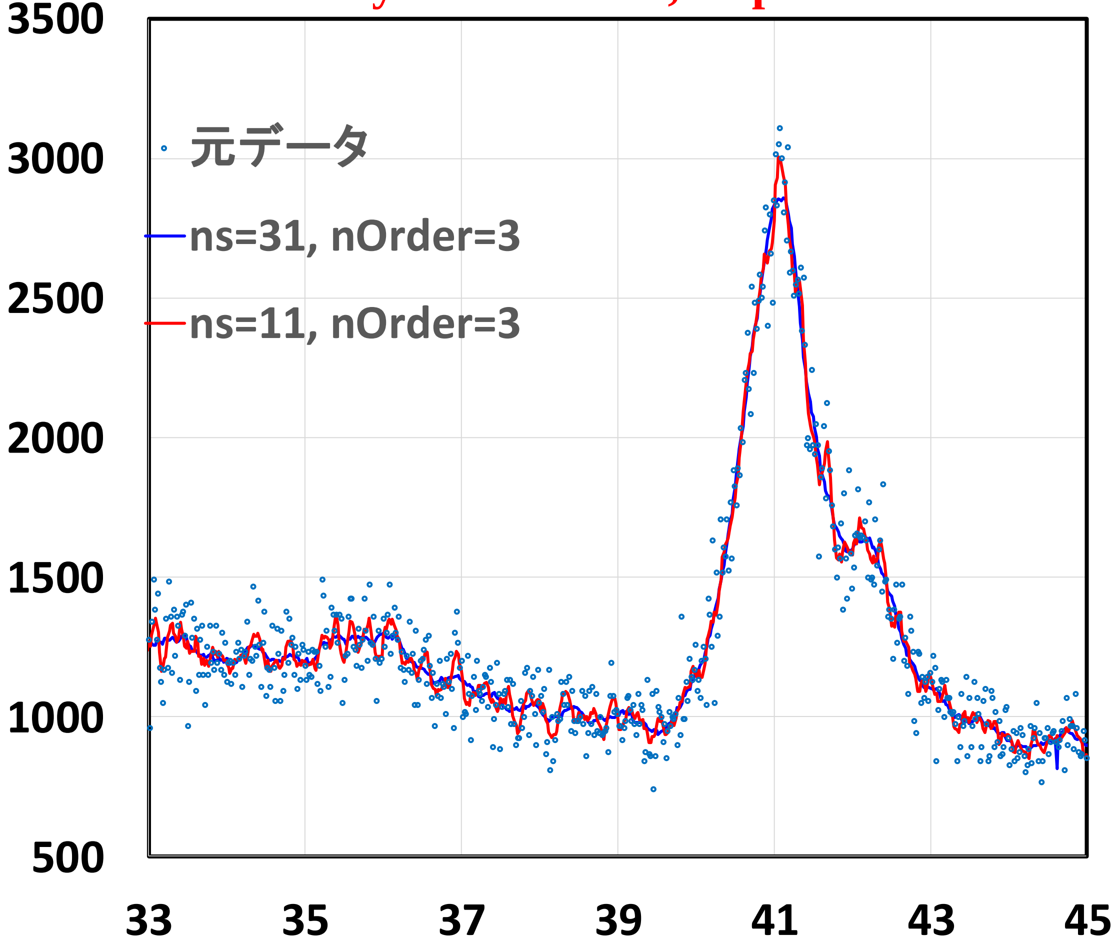


Smoothing: Polynomial fit method (多項式適合法)

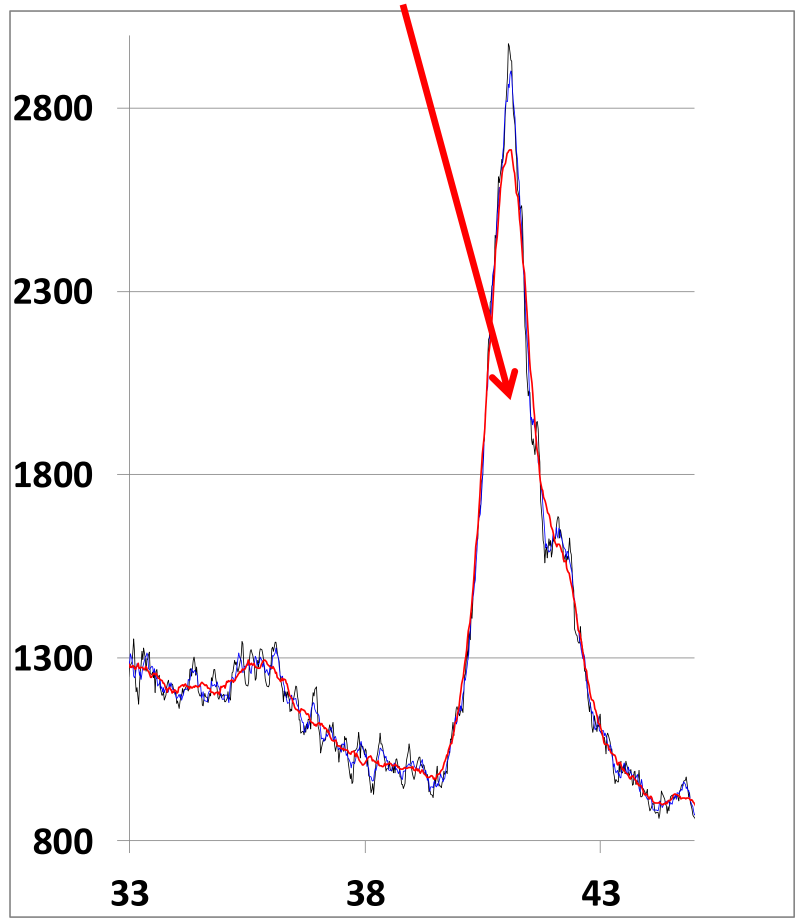
Adopt n_{order} order polynomial to n_s sample points among the given n sample points, determined by LSQ

データに n_{order}次多項式を最小自乗法で求め、標本点の値を内挿する

Polynomial fit: 11, 31 points



Simple moving average: 5, 11, 31 points



Weights of polynomial fit (Savizky-Golay method)

多項式適合法 (Savizky-Golay法) の重み

南茂夫, 科学計測のための波形データ処理, CQ出版社 (1986)

Table 5.1 Weights for order 2 and 3 polynomial fit

# of points N	25	23	21	19	17	15	13	11	9	7	5
m=int(N/2)	12	11	10	9	8	7	6	5	4	3	2
-12	-253										
-11	-138	-210									
-10	-33	-105	-171								
-9	62	-10	-76	-136							
-8	147	75	9	-51	-105						
-7	222	150	84	24	-30	-78					
-6	287	215	149	89	35	-13	-55				
-5	342	270	204	144	90	42	0	-36			
-4	387	315	249	189	135	87	45	9	-21		
-3	422	350	284	224	170	122	80	44	14	-10	
-2	447	375	309	249	195	147	105	69	39	15	-3
-1	462	390	324	264	210	162	120	84	54	30	12
0	467	395	329	269	215	167	125	89	59	35	17
1	462	390	324	264	210	162	120	84	54	30	12
2	447	375	309	249	195	147	105	69	39	15	-3
3	422	350	284	224	170	122	80	44	14	-10	
4	387	315	249	189	135	87	45	9	-21		
5	342	270	204	144	90	42	0	-36			
6	287	215	149	89	35	-13	-55				
7	222	150	84	24	-30	-78					
8	147	75	9	-51	-105						
9	62	-10	-76	-136							
10	-33	-105	-171								
11	-138	-210									
12	-253										
Normalization factor	**5175**	**4025**	**3059**	**2261**	**1615**	**1105**	**715**	**429**	**231**	**105**	**35**

Order 1: Simple moving average
Orders 2 and 3 have the same weights

Weights for order 2 and 3 using (2*m*+1) points ((2*m*+1)点を用いた2,3次多項式適合の重み)

$$w_{23}(j) = 3m(m+1) - 1 - 5j^2 \qquad j = -m, \cdots, -1, 0, 1, \cdots, m$$

$$W_{23} = (4m^2 - 1)(2m + 3)/3$$

Calculation

Simple moving average (2m+1 points)

$$y_{i,smoothed} = \frac{1}{2m+1} \sum_{j=i-m}^{i+m} y_j$$

Weighted moving average (2m+1 points)

$$y_{i,smoothed} = \sum_{j=i-m}^{i+m} w_i y_j \,/\, \sum_{j=i-m}^{i+m} w_i$$

Order 2 and 3 polynomial fit using (2*m*+1) points

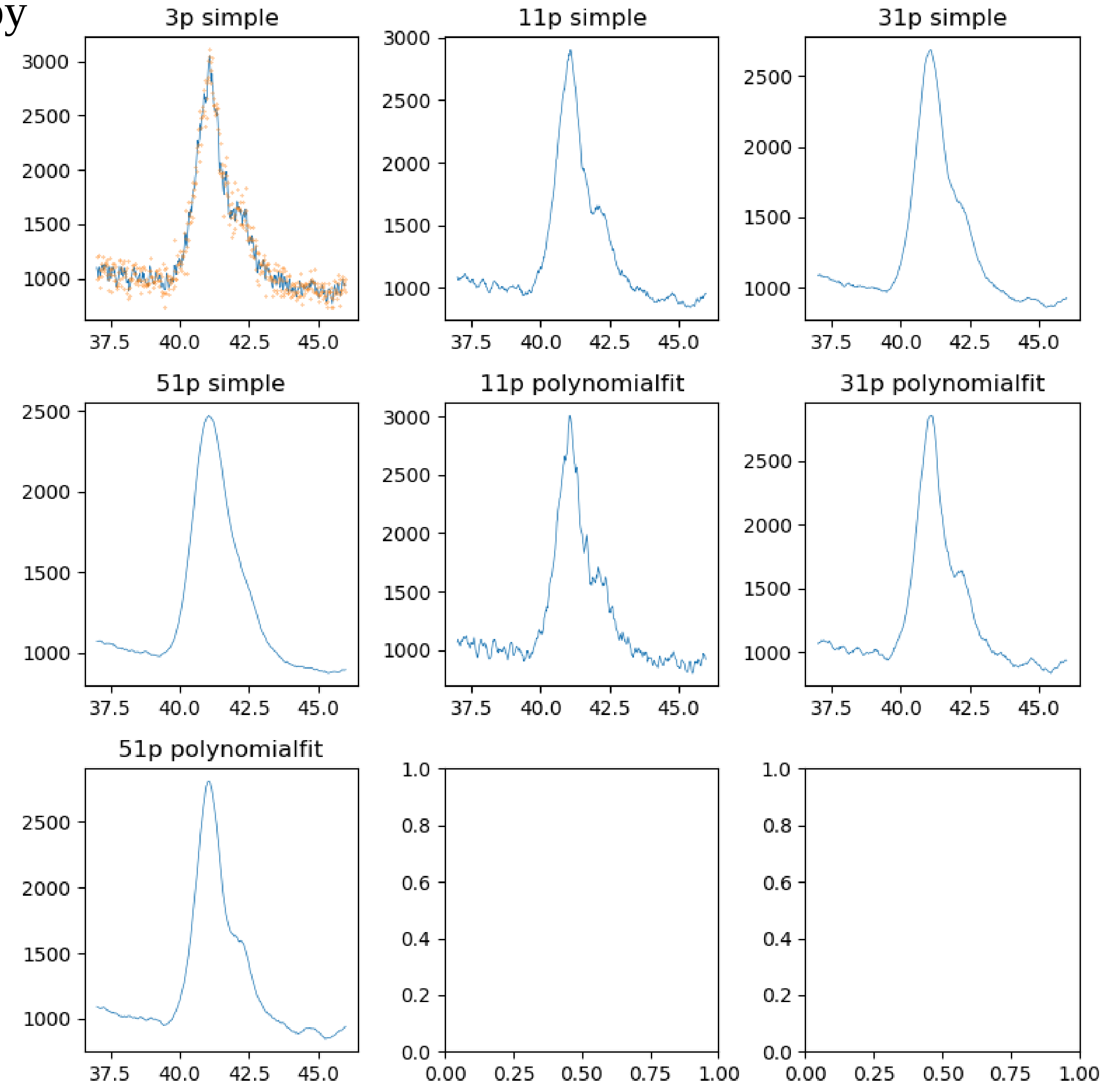
$w_{23}(j) = 3m(m+1) - 1 - 5j^2$ j = -m, ⋯, -1, 0, 1, ⋯, m

$W_{23} = (4m^2 - 1)(2m + 3)/3$

$$y_{i,smoothed} = \frac{1}{W_{23}} \sum_{j=i-m}^{i+m} w_{23}(j) y_j$$

Program: smoothing.py

Usage: python smoothing.py



Fourier transformation (フーリエ変換)

Different definitions

FT (フーリエ変換) $$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(i\omega t)dt$$

IFT (逆フーリエ変換) $$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(-i\omega t)d\omega$$

FT $$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(i2\pi ft)dt$$

IFT $$f(t) = \int_{-\infty}^{\infty} F(\omega)\exp(-i2\pi ft)d\omega$$

Features of Fourier transformation

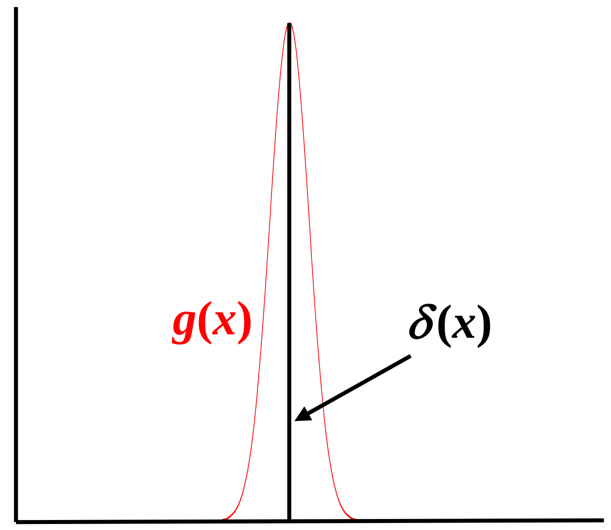
- **Convert time-dependent data to frequency data**
- **Convert position-dependent data to wavenumber data**
- **Origin of original data is converted to whole range of FT data**
- **Whole range of original data is converted to origin of FT data**
- **IFT of FTed data recovers the original data**

Fourier変換したデータをFourier逆変換すると元のデータに戻る

Convolution (畳み込み)

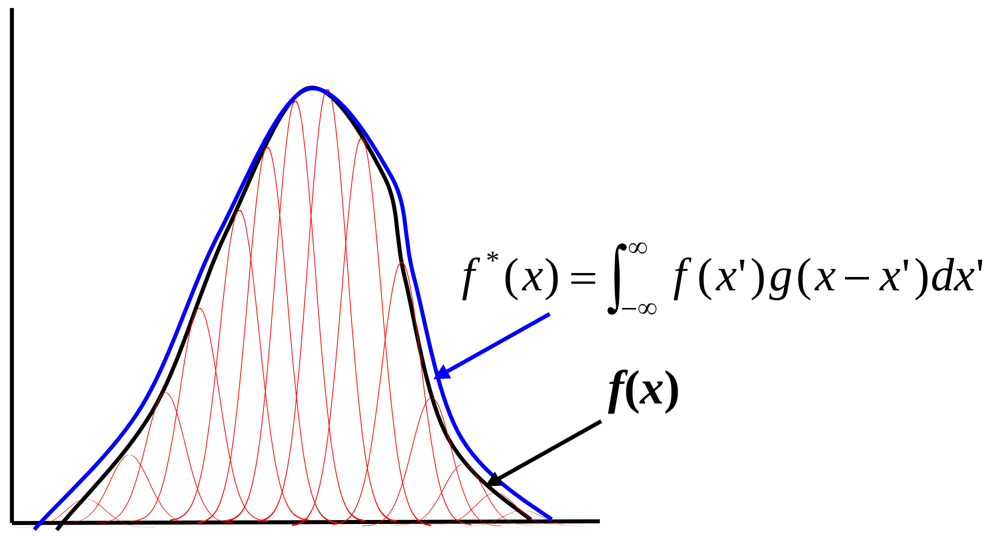
$$(f * g)(x) = f^*(x) = \int_{-\infty}^{\infty} f(x')g(x - x')dx'$$

Observed peak has a finite width originating from apparatus function *g*(*x*)
Even if the intrinsic peak has zero width (delta function δ(*x*))
試料本来のデータは線幅ゼロ (δ関数) でも、
測定値は装置関数 *g*(*x*) の広がりを持つ



$$\int_{-\infty}^{\infty} g(x)dx = 1$$

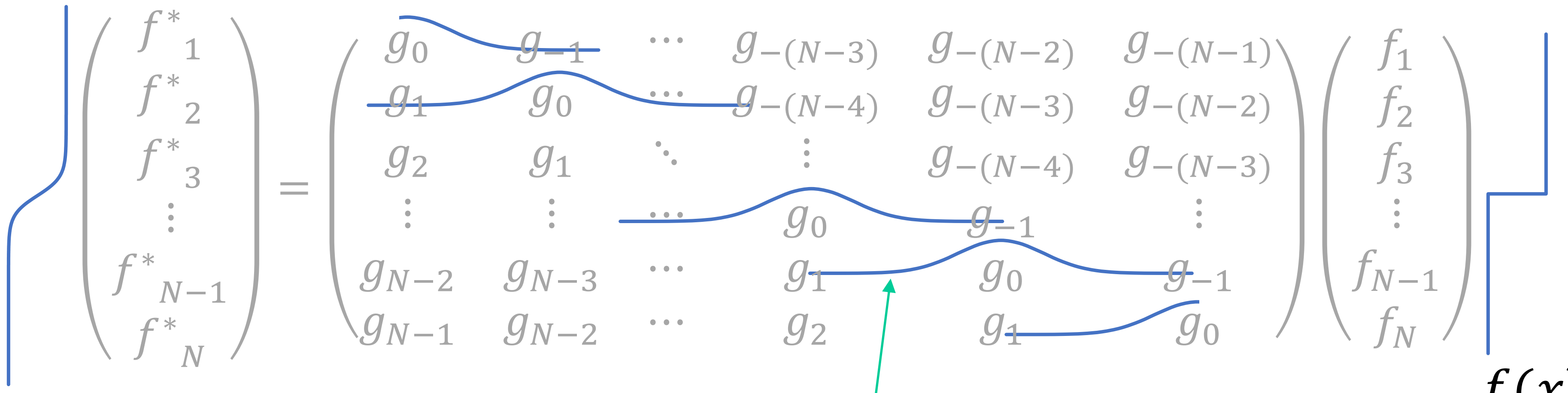
For a real case a sample has an intrinsic peak *f*(*x*), the observed peak will be a convolution of *f*(*x*) and apparatus function *g*(*x*), *f(*x*).**
試料本来のデータは *f*(*x*) でも、測定されるのは
装置関数 *g*(*x*) の畳み込みをした $f^*(x)$



Convolution: Matrix representation (行列表示)

南茂夫 編著、科学計測のための波形データ処理、CQ出版 (1986年)

$$f^*(x_i) = \int_{-\infty}^{\infty} f(x')g(x_i - x')dx' = N^{-1}\sum_{j=1}^{N} f(x_j)g(x_i - x_j)$$



$$\begin{pmatrix} f^*{}_1 \\ f^*{}_2 \\ f^*{}_3 \\ \vdots \\ f^*{}_{N-1} \\ f^*{}_N \end{pmatrix} = \begin{pmatrix} g_0 & g_{-1} & \cdots & g_{-(N-3)} & g_{-(N-2)} & g_{-(N-1)} \\ g_1 & g_0 & \cdots & g_{-(N-4)} & g_{-(N-3)} & g_{-(N-2)} \\ g_2 & g_1 & \ddots & \vdots & g_{-(N-4)} & g_{-(N-3)} \\ \vdots & \vdots & \cdots & g_0 & g_{-1} & \vdots \\ g_{N-2} & g_{N-3} & \cdots & g_1 & g_0 & g_{-1} \\ g_{N-1} & g_{N-2} & \cdots & g_2 & g_1 & g_0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N \end{pmatrix}$$

$f^*(x)$
Observed signal

$g(x_i - x_j)$
Apparatus function

$f(x)$
Intrinsic signal

Very often, matrix g_{ij} is band matrix with maxima at diagonal

(行列g_{ij}は対角要素に最大値を持つ帯行列になることが多い)

Smoothing by convolution (smearing)

畳み込みによる平滑化 (ぼかし)

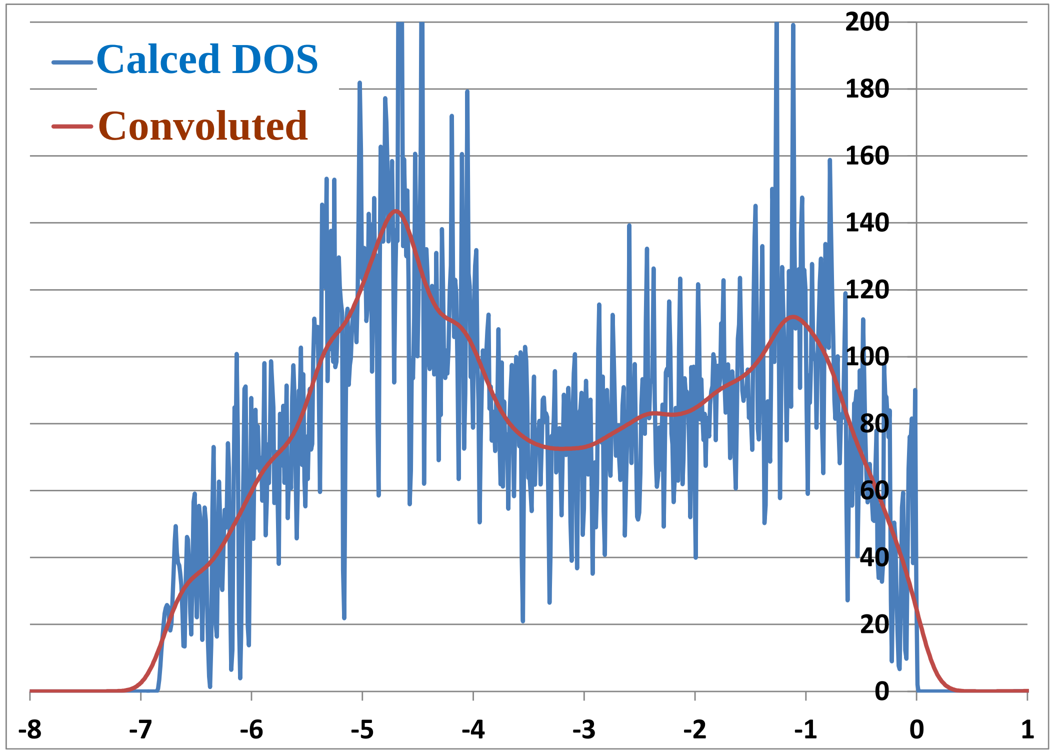
Density of state (DOS) function calculated by density functional theory

密度汎関数計算で得たa-$InGaZnO_4$の状態密度

Problem: Many noise, difficult to read

Add finite-width Gauss function to each data (それぞれのデータにGauss関数の広がり)

$G(E) = \exp(-[(E - E_0)/w]^2)$ (w = 0.2 eV)



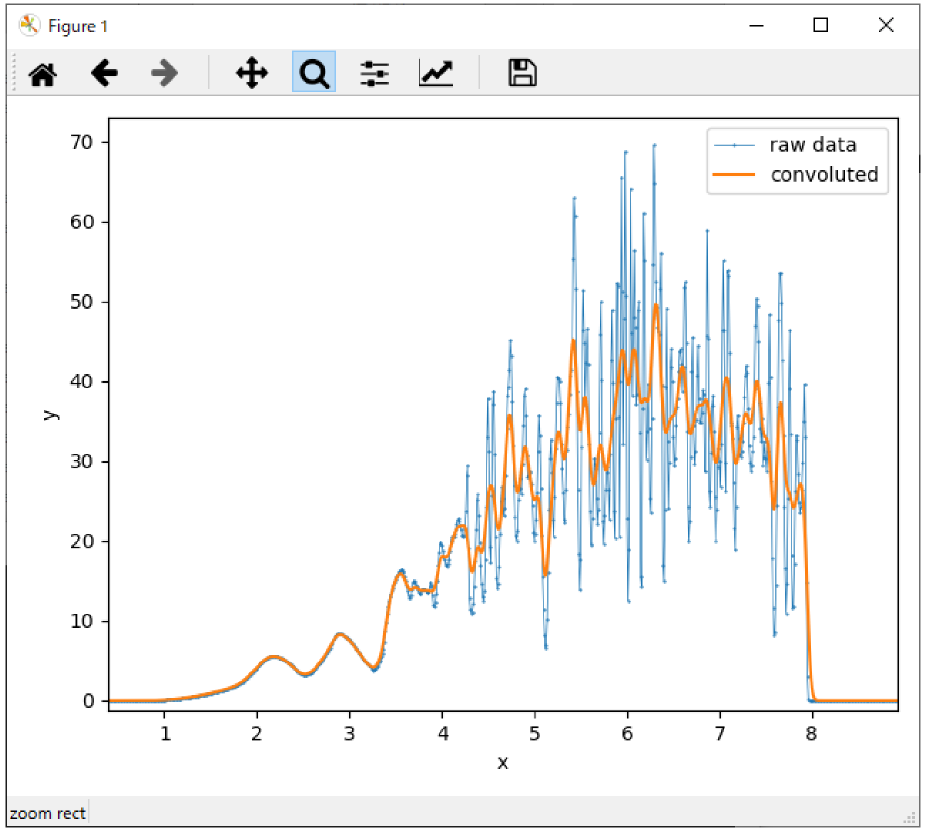
Note: Estimation of band, edge energies will have the errors originating from the smearing width *w*

Program: convolution.py

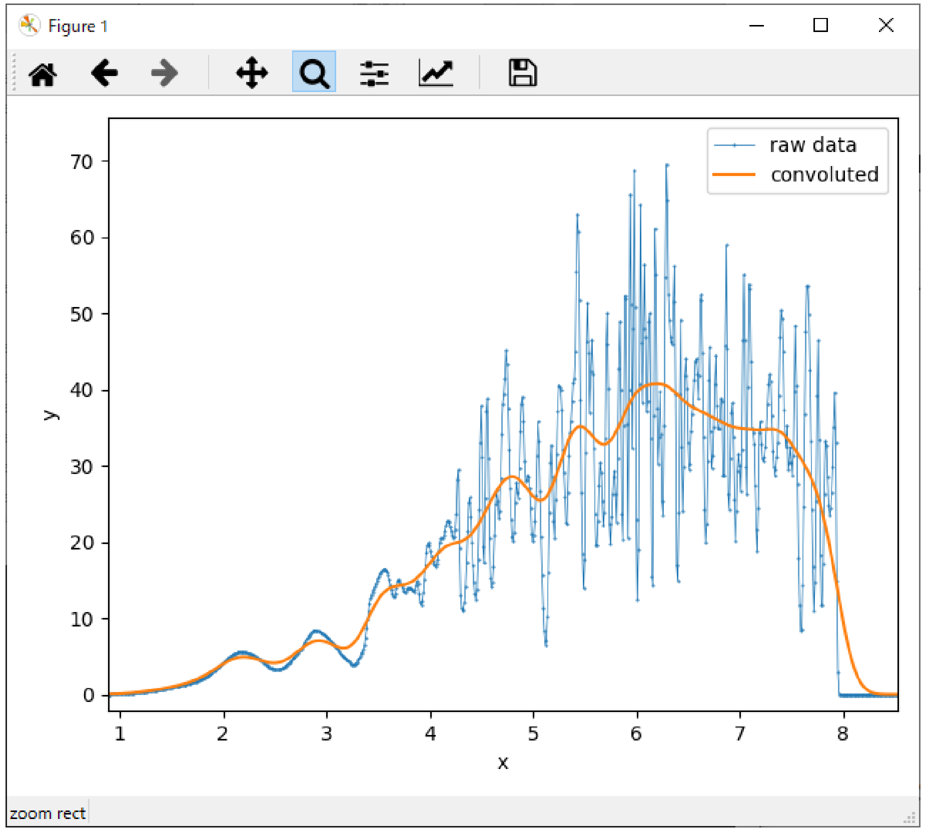
Usage: python convolution.py width

width: width of Gaussian function to convolute

python convolution.py 0.05



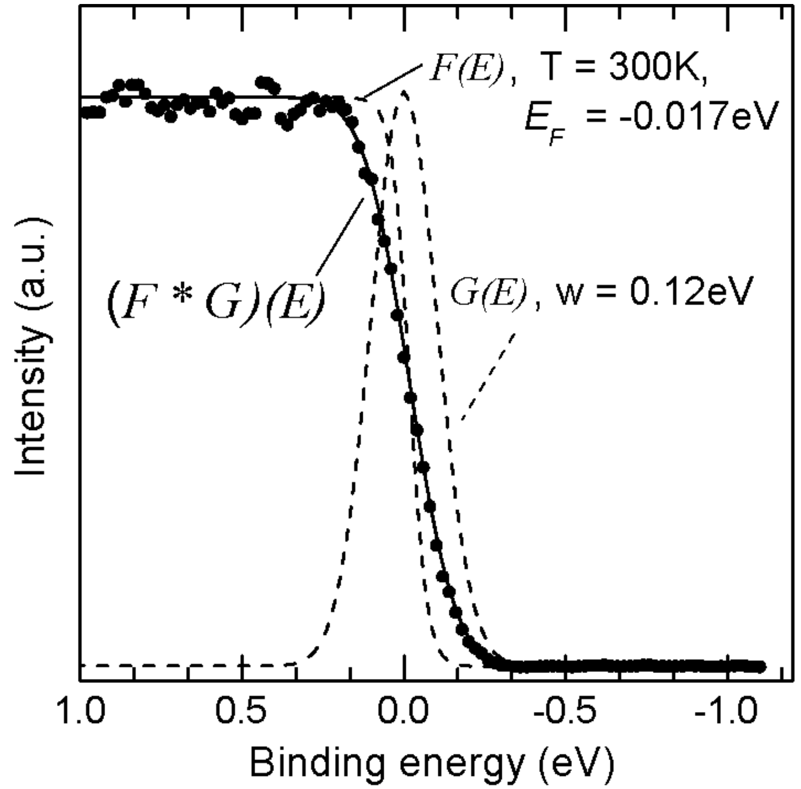
python convolution.py 0.2



Convolution (畳み込み)

$$(f * g)(x) = f^*(x) = \int_{-\infty}^{\infty} f(x')g(x-x')dx'$$

Example: UPS spectrum of Au Fermi edge



Intrinsic sample spectrum

$S(E)$

Apparatus function

$G(E) = G_0\exp(-[(E-E_0)/aw]^2)$

Fermi-Dirac distribution

$f(E) = 1/(1+\exp[(E-E_F)/k_BT])$ eq. (1)

Observed spectrum

$$I(x) = \int_{-\infty}^{\infty} S(E')G(E-E')f(E-E')dE'$$

Assuming constant $S(E)$ for Au reference,

$G(E)$ is determined by fitting eq. (1) to I(x)

w = 0.12 eV

$S(E)$ for different sample is obtained by deconvolution using the $G(E)$ obtained by the reference spectrum

$G(E)$がわかると、他の実測スペクトルから 逆畳み込みで$S(E)$ がわかる

Filter and convolution

Convolution:
Matrix product of data vector and filter

First differential

$$\frac{dy}{dx_2} = \boxed{\underset{\text{Filter}}{\frac{1}{2h}\begin{pmatrix}-1 & 0 & 1\end{pmatrix}}}\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}$$

Second differential

$$\frac{d^2y}{dx^2{}_2} = \frac{1}{2h^2}\begin{pmatrix}1 & -2 & 1\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}$$

Simple moving average (3 points)

$$y_{2,s} = \frac{1}{3}\begin{pmatrix}1 & 1 & 1\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}$$

Weighted moving average (3p one-side triangle)

$$y_{2,s} = \frac{1}{3}\begin{pmatrix}0 & 2 & 1\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}$$

Weighted moving average (3p double-side triangle)

$$y_{2,s} = \frac{1}{4}\begin{pmatrix}1 & 2 & 1\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}$$

Polynomial fit smoothing (2*m*+1 points)

$$y_{3,s} = \frac{1}{35}\begin{pmatrix}-3 & 12 & 17 & 12 & -3\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\\y_4\\y_5\end{pmatrix}$$

Differentiation, smoothing, convolution may be performed with the same convolution program by adopting appropriate filters.

微分、平滑化、コンボリューションは、 フィルターを変えるだけで 同じコンボリューションプログラムを流用できる

Smoothing by python library

Smoothing. py

Simple moving average

```
# make a constant filter with weight of 1/N
    w = np.ones(nsmooth) / nsmooth
# convolution
ys = np.convolve(y, w, mode = ‘same‘)
```

Polynomial smoothing: **Savizky-Golay filter**

```
   ys = scipy.signal.savgol_filter(y, nsmooth, norder, deriv = 0)
```

nsmooth: number of smoothing points

norder : order of polynomial

deriv : order of differentiation

注意: savgol_filter() takes differentiation to specify deriv = 1, but its absolute values are different from the first differentials because savgol_filter() does not know x-axis values.

Divide the results by h^{deriv} after smoothing if you need absolute values

Note: check the final results by yourself

Multi-dimensional filter, convolution, image processing

filter: $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ $\quad y'_{22} = \sum_{i,j=1,2,3} a_{ij} y_{ij}$

e.g., the convolution of y_{22} is given by a sum of each product of

data $\begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{pmatrix}$ **and**

filter $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$

NOTE: different from convolution in mathematics

X differential filter (edge detection) $\frac{1}{2h}\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$

Y differential filter (edge detection) $\frac{1}{2h}\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

Diagonal differential filter (edge detection) $\frac{1}{2h}\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$

Smoothing filter (smearing/blur) **⇔ sharpening can be done by deconvolution**

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Convoluted Neural Network (畳み込みニューラルネットワーク)

Insert a convolution layer in multilayer neural network to learn the elements of the filter

We may know how the filter works by analyzing the values of the filter elements

Deconvolution (逆畳み込み)

$$(f*g)(x) = f^*(x) = \int_{-\infty}^{\infty} f(x')g(x-x')dx'$$

Fourier transformation (FT) $F^*(k) = \int_{-\infty}^{\infty} f^*(x)\exp(ikx)dx$

Inverse Fourier transformation (IFT) $f(x) = \int_{-\infty}^{\infty} F(k)\exp(-ikx)dk$

$g(x) = \int_{-\infty}^{\infty} G(k')\exp(-ik'x)dk'$

$$\begin{aligned}
F^*(k) &= \int_{-\infty}^{\infty} f(x)g(x-x')\exp(ikx)dxdx' \\
&= \int_{-\infty}^{\infty} f(x)\left(\int g(x-x')\exp(ikx)dx\right)dx' \\
&= \int_{-\infty}^{\infty} f(x)\left(\int g(x)\exp(ik(x+x'))dx\right)dx' \\
&= \int_{-\infty}^{\infty} f(x)G(k)\exp(ikx')dx' \\
&= F(k)G(k)
\end{aligned}$$

***f*(*x*) can be obtained by IFT of *F*(*k*) = *F**(*k*) / *G*(*k*), but usually is vulnerable against small perturbations like noise**

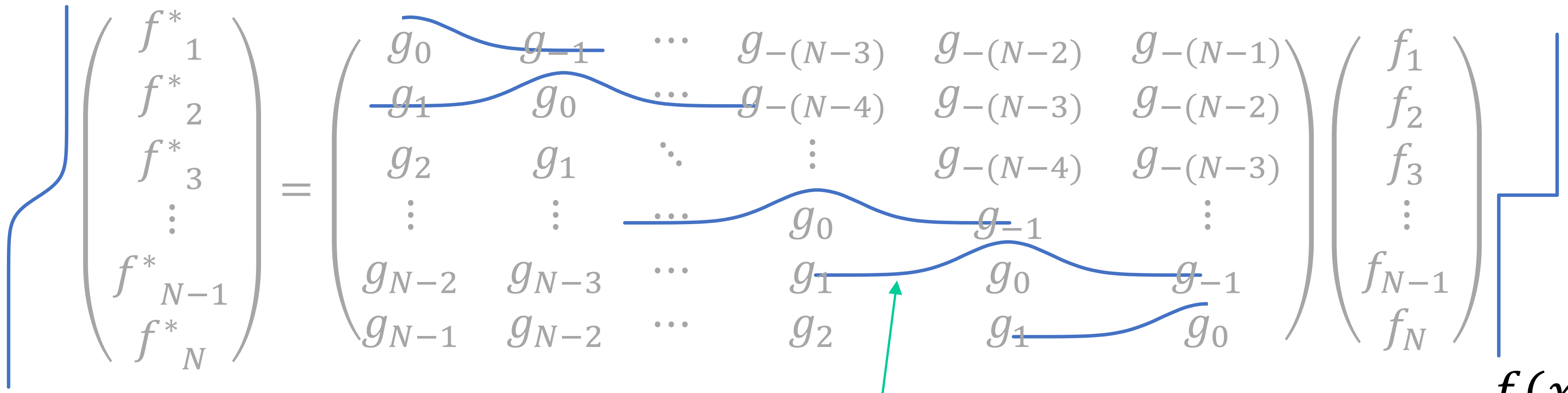
$F(k) = F^*(k) / G(k)$を計算して逆フーリエ変換で $f(x)$ が得られる

=> ノイズなどがあると不安定で解が発散しやすい

Convolution: Matrix representation (行列表示)

南茂夫 編著、科学計測のための波形データ処理、CQ出版 (1986年)

$$f^*(x_i) = \int_{-\infty}^{\infty} f(x')g(x_i - x')dx' = N^{-1}\sum_{j=1}^{N} f(x_j)g(x_i - x_j)$$



$$\begin{pmatrix} f^*{}_1 \\ f^*{}_2 \\ f^*{}_3 \\ \vdots \\ f^*{}_{N-1} \\ f^*{}_N \end{pmatrix} = \begin{pmatrix} g_0 & g_{-1} & \cdots & g_{-(N-3)} & g_{-(N-2)} & g_{-(N-1)} \\ g_1 & g_0 & \cdots & g_{-(N-4)} & g_{-(N-3)} & g_{-(N-2)} \\ g_2 & g_1 & \ddots & \vdots & g_{-(N-4)} & g_{-(N-3)} \\ \vdots & \vdots & \cdots & g_0 & g_{-1} & \vdots \\ g_{N-2} & g_{N-3} & \cdots & g_1 & g_0 & g_{-1} \\ g_{N-1} & g_{N-2} & \cdots & g_2 & g_1 & g_0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N \end{pmatrix}$$

$f^*(x)$
Observed signal

$g(x_i - x_j)$
Apparatus function

$f(x)$
Intrinsic signal

Very often, matrix g_{ij} is band matrix with maxima at diagonal

(行列g_{ij}は対角要素に最大値を持つ帯行列になることが多い)

Deconvolution (逆畳み込み)

南茂夫 編著、科学計測のための波形データ処理、CQ出版 (1986年)

$$f^*(x_i) = N^{-1}\sum_{j=1}^{N} f(x_j)g(x_i - x_j)$$

Deconvolution is carried out by solving the linear simultaneous equations,

$$\begin{pmatrix} f^*{}_1 \\ f^*{}_2 \\ \vdots \\ f^*{}_N \end{pmatrix} = \begin{pmatrix} g_0 & g_{-1} & & g_{-(N-1)} \\ g_1 & g_0 & & \\ \vdots & & \ddots & \vdots \\ g_{N-1} & & \cdots & g_0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

However, similar to the FFT method, usually vulnerable against noise
(フーリエ変換法と同様、ノイズなどがあると不安定で解が発散しやすい)

Better way:

1. **Remove noise effects (smoothing etc) before deconvolution**
2. **Use an iterative method (e.g., Jacobi method and Gauss-Seidel method) to solve the simultaneous equation, where noise-compensation process is included during the iteration process.**

Jacobi / Gauss-Seidel method

Solve large-size simultaneous linear equations:

$$\begin{pmatrix} a_{11} & a_{12} & & a_{1N} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{N1} & & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

For (*k*+1)-th iteration, $\boldsymbol{x_i}^{(k+1)}$ is estimated from $\boldsymbol{x_i}^{(k)}$

(initial data may be chosen as $x_i^{(0)} = b_i$, uniform value $x_i^{(0)} = 1$, etc):

(i) Jacobi method: $\boldsymbol{x_i^{(k+1)} = \left(b_i - \sum_{j\neq i}^{N} a_{ij} x_j^{(k)}\right)/a_{ii}}$

$$x_1^{(k+1)} = \left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1N}x_N^{(k)}\right)/a_{11}$$
$$x_2^{(k+1)} = \left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2N}x_N^{(k)}\right)/a_{22}$$

(ii) Gauss-Seidel method: $\boldsymbol{x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(k)}\right)/a_{ii}}$

Using the known $\boldsymbol{x_j}^{(k+1)}$ values enhances convergence.

$$x_1^{(k+1)} = \left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1N}x_N^{(k)}\right)/a_{11}$$
$$x_2^{(k+1)} = \left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2N}x_N^{(k)}\right)/a_{22}$$

Convergence is better for the Gauss-Seidel method,
While parallelization is more easy for the Jacobi method.

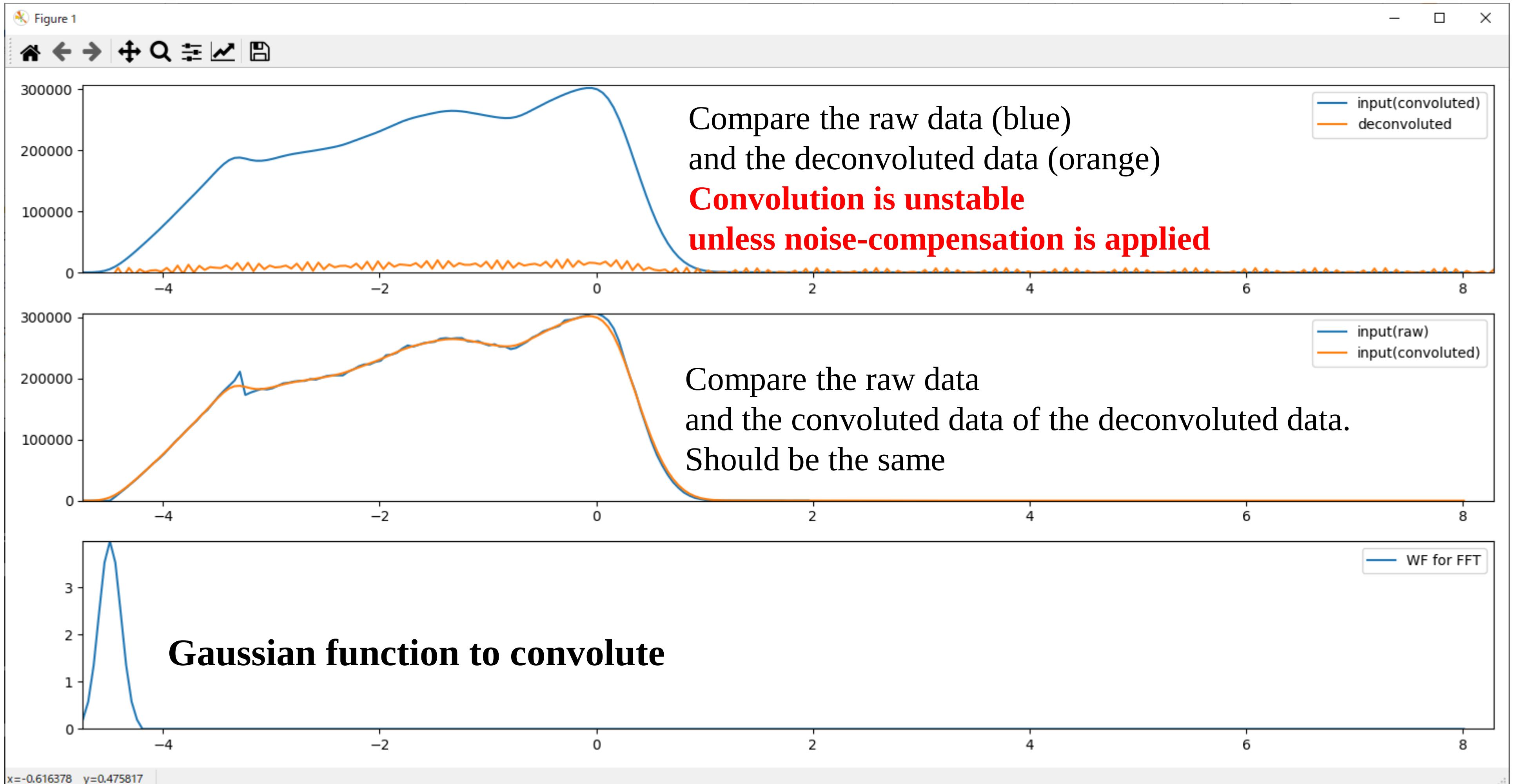
Program: deconvolution.py

Usage: python deconvolution.py file mode convmode smoothmode xmin xmax Wa Grange kzero klin

see usage of the program output

```
python deconvolution.py pes.csv fft full convolve+extend -4.5 2.0 0.12 2.0 5 5
```

Use **FFT and iFFT without smoothing**



Compare the raw data (blue)
and the deconvoluted data (orange)
Convolution is unstable
unless noise-compensation is applied

Compare the raw data
and the convoluted data of the deconvoluted data.
Should be the same

Gaussian function to convolute

Deconvolution: Gauss-Seidel method w/o smooting

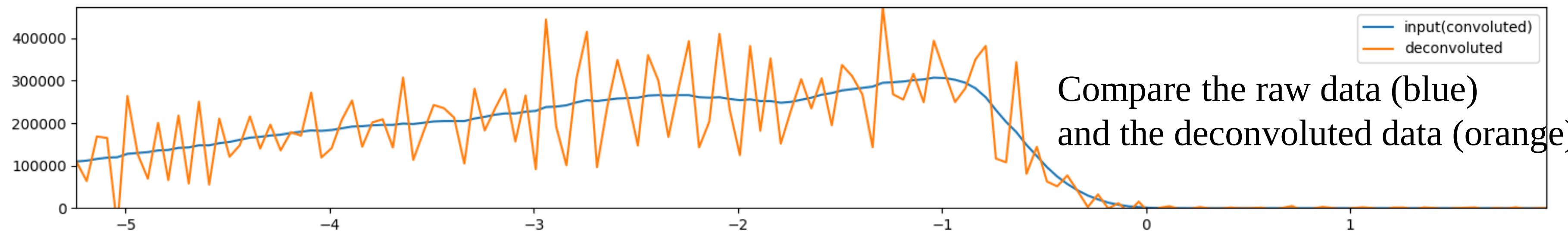
Usage: python deconvolution.py file mode xmin xmax Wa dump nmaxiter eps nsmooth zeroc

see usage of the program output

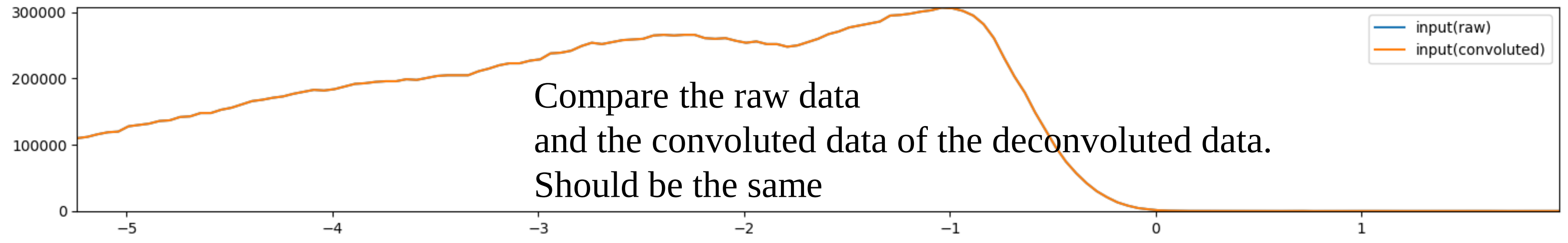
```
python deconvolution.py pes.csv gs -6.0 2.0 0.12 1.0 300 1.0e-4 1 0
```

Use **Gauss-Seidel (gs) method** with the width of the Gaussian function of 0.12 eV.

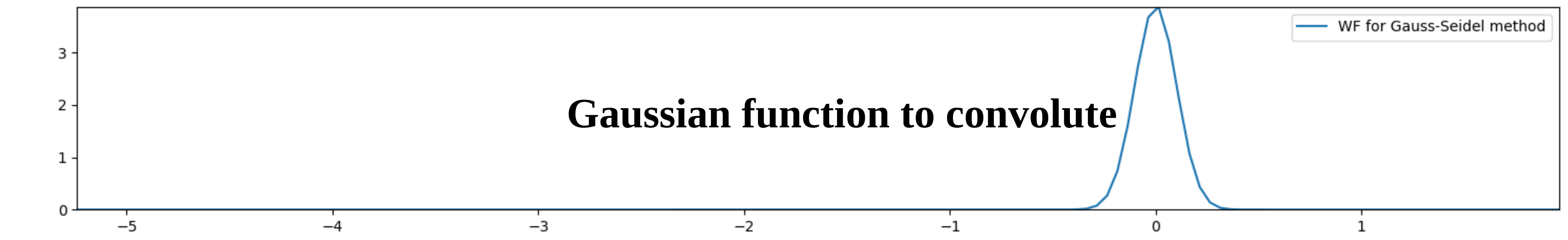
No smoothing is applied for each iteration.



Compare the raw data (blue)
and the deconvoluted data (orange)



Compare the raw data
and the convoluted data of the deconvoluted data.
Should be the same



Gaussian function to convolute

Program: Gauss-Seidel method with smoothing

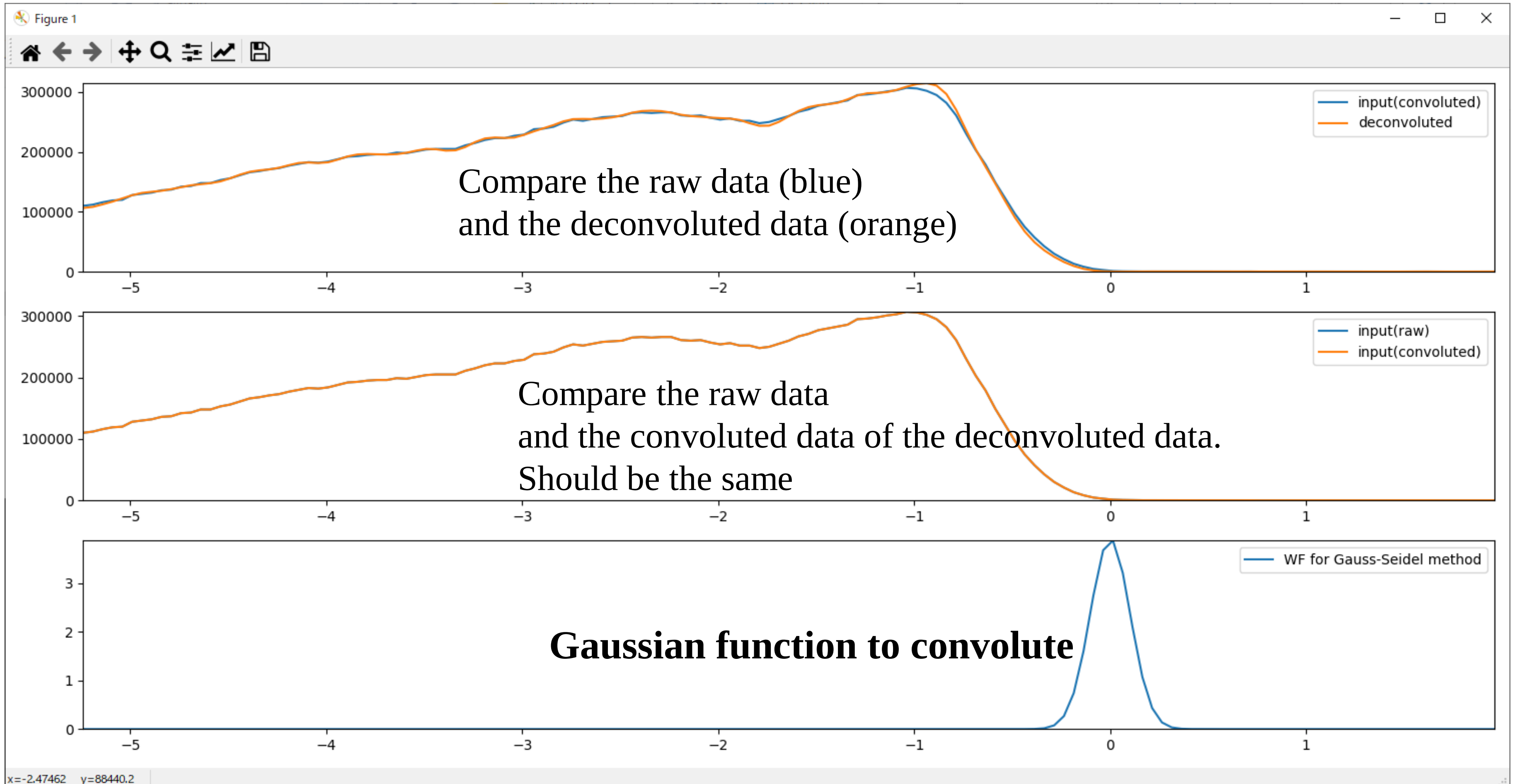
Usage: python deconvolution.py file mode xmin xmax Wa dump nmaxiter eps nsmooth zeroc

see usage of the program output

```
python deconvolution.py pes.csv gs -6.0 2.0 0.12 1.0 300 1.0e-4 5 0
```

Use **Gauss-Seidel (gs) method** with the width of the Gaussian function of 0.12 eV.

5-point polynomial-fit average is applied for each iteration.



Compare the raw data (blue)
and the deconvoluted data (orange)

Compare the raw data
and the convoluted data of the deconvoluted data.
Should be the same

Gaussian function to convolute

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)**)**

Mini max approximation: minimize $\max\limits_{a \le x \le b} |f(x_i) - y_i|$

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

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

$$S = \Sigma(a + bx_i - y_i)^2$$

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

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

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

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

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

Mini-max approximation

入門 数値計算

Minimize $\max_{a \le x \le b} |f(x_i) - y_i|$

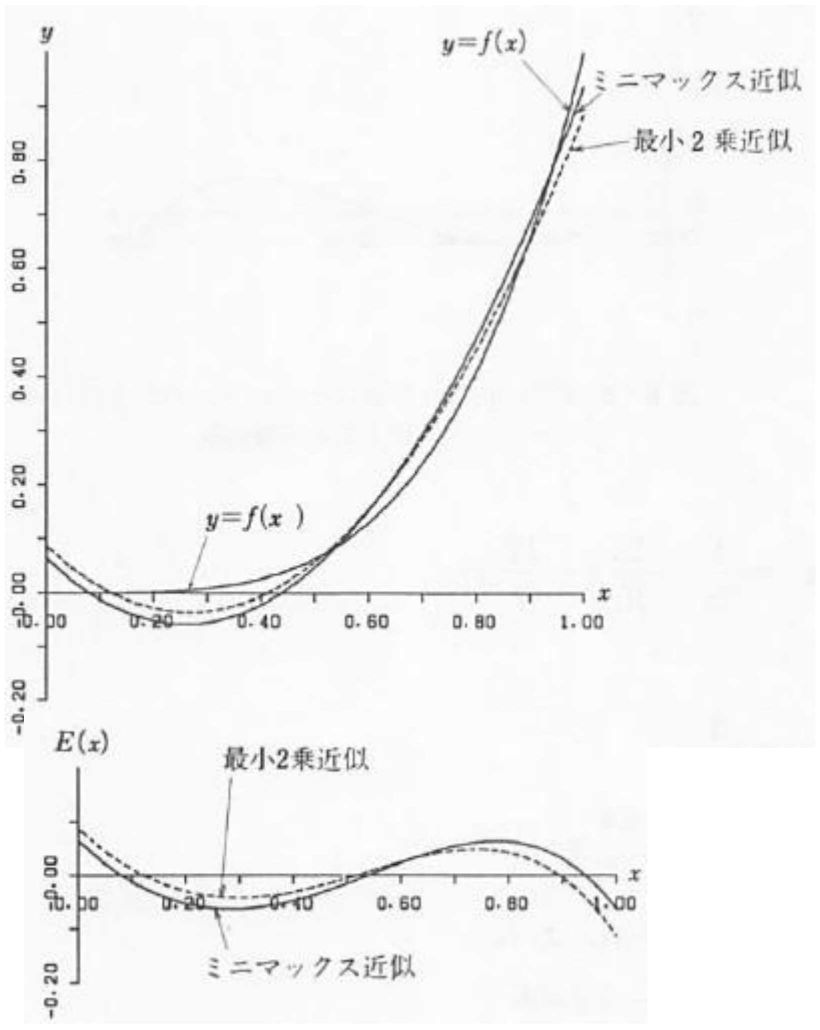


図 6・3 ミニマックス近似と最小2乗近似

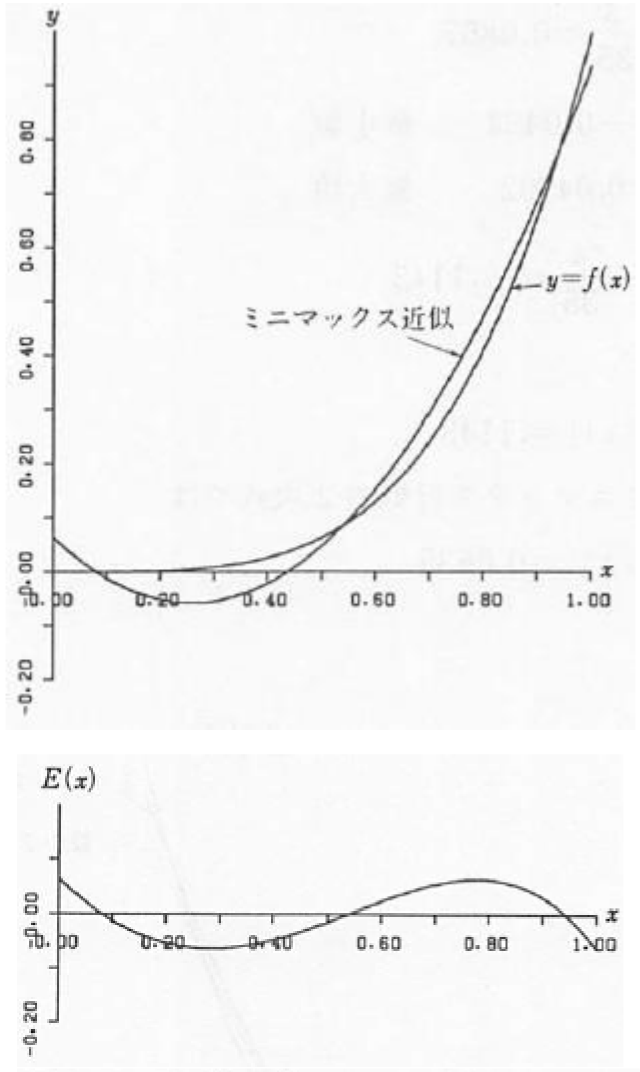


図 6・2 区間 [0, 1] における $f(x)=x^4$ の2次のミニマックス近似とその誤差曲線

*l*LSQ: Polynomial

線形最小二乗法: 多項式

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

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

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

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

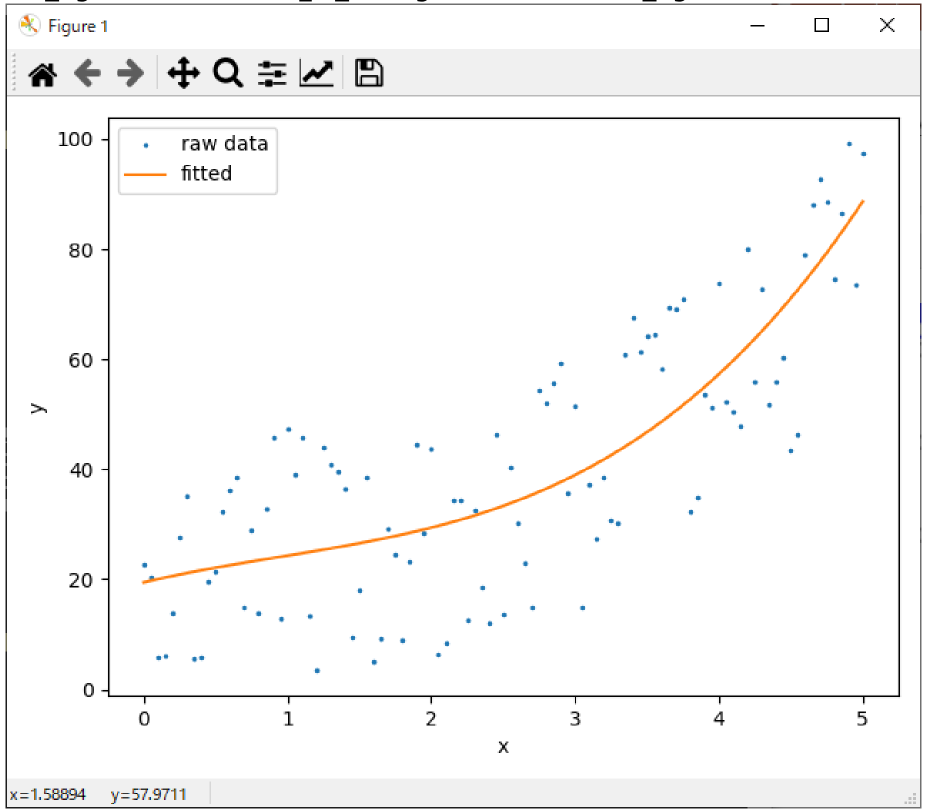
$|x_i| > 1$ might cause overflow,
$|x_i| < 1$ might cause underflow errors.
=> Normalize the *x* range e.g. to [-1, 1] : $x_i' = 2\frac{x_i - x_{\mathrm{mid}}}{x_{\mathrm{max}} - x_{\mathrm{min}}}$

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

Program: lsq-polynomial.py

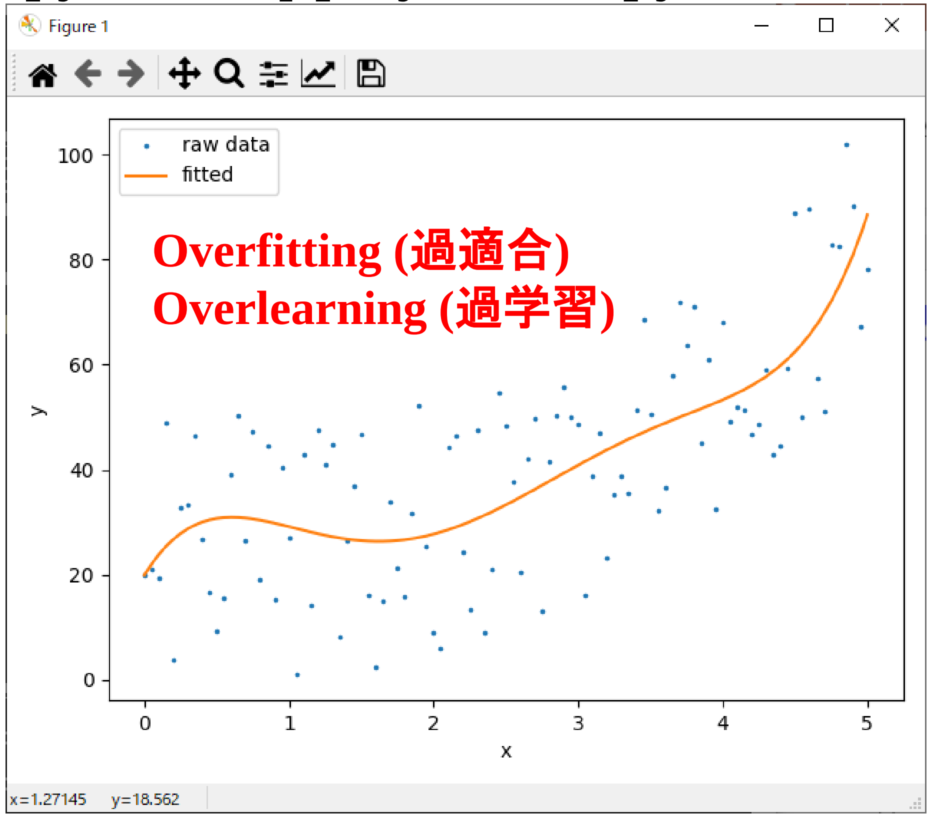
Usage: python lsq-polynomial.py n_{order}

python lsq-polynomial.py **3**



```
x=1.58894    y=57.9711
```

python lsq-polynomial.py **6**



Overfitting (過適合)
Overlearning (過学習)

```
x=1.27145    y=18.562
```

*l*LSQ: General functions

線形最小二乗法: 一般関数の場合

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

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

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

If *f(x)* is *linear with respect to fitting parameters*,
final solution is obtained by one matrix operation

係数に関して線形であれば、1度の行列計算で最終解が得られる

ex. $f(x) = a + b\log x + c/x$

$$f(x, y) = a + bxy + cy/x$$

Program: lsq-general.py

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

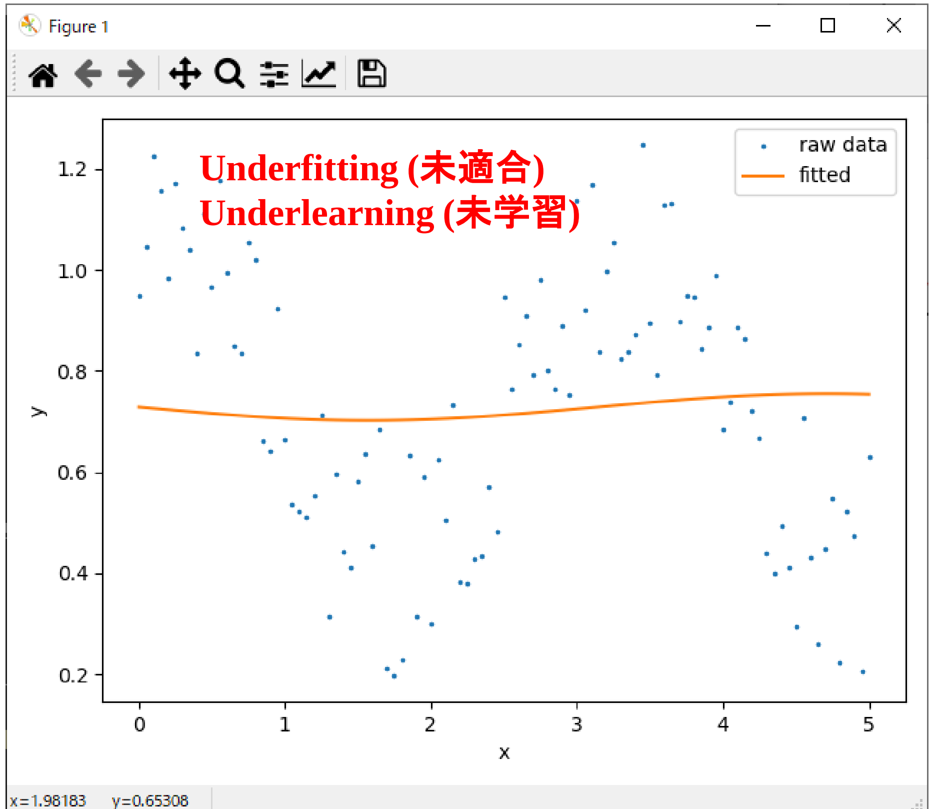
fit to $y = c_0 + c_1 \sin x + c_2 \cos x + c_3 \sin 2x + c_4 \cos 2x$
$+ c_5 \sin 3x + c_6 \cos 3x$

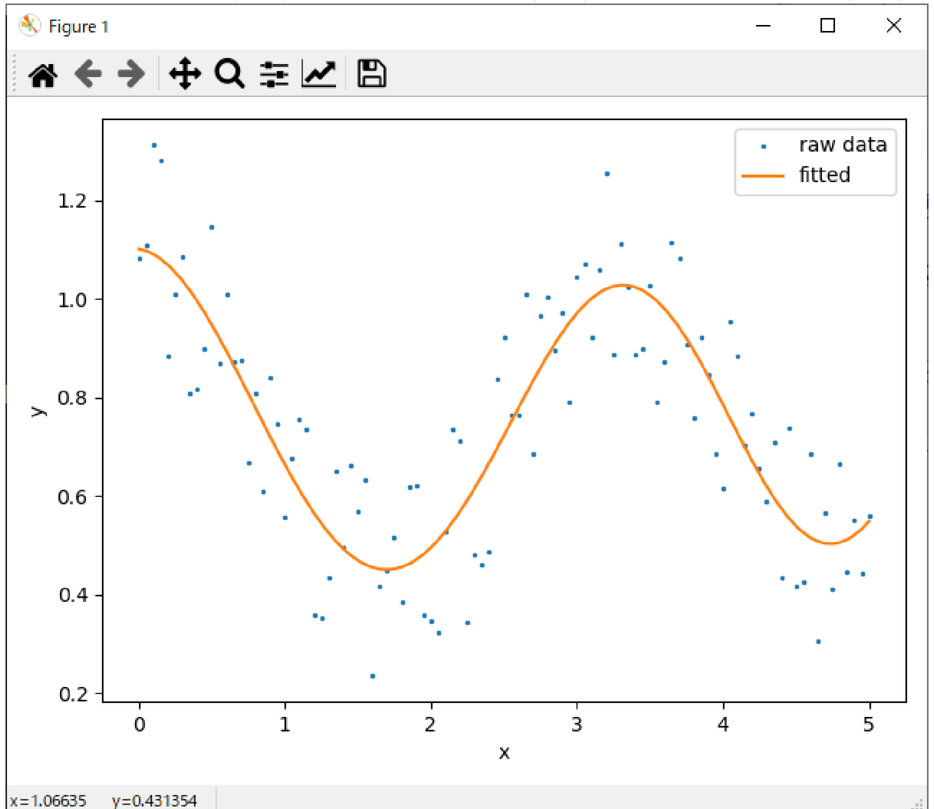
python lsq-general.py 2

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) + 0.303cos(2x) + 0.0119sin(3x)





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

(三斜晶結晶の面間隔)

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

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

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

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

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

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

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

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

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

The form of $d_{\mathrm{hkl}}{}^{-2}$ 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)