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)

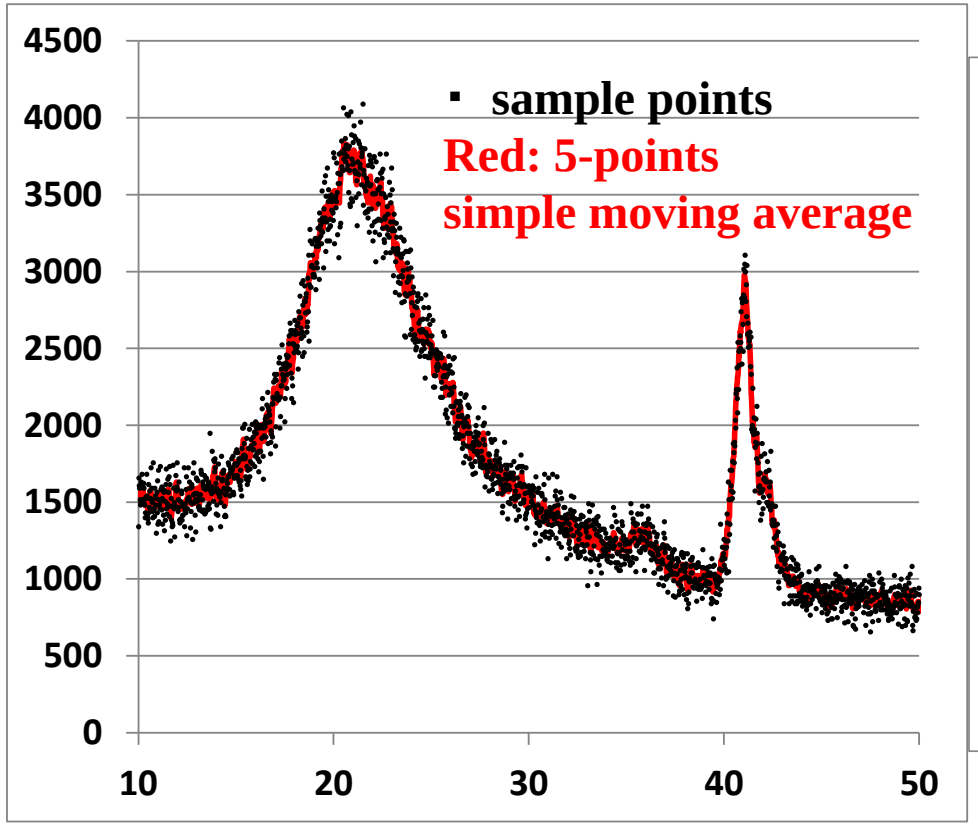
$$y_{i,smoothed} = \sum_{j=i-m}^{i+m} w_j y_j \; / \sum_{j=i-m}^{i+m} w_i$$

Smoothing (平滑化)

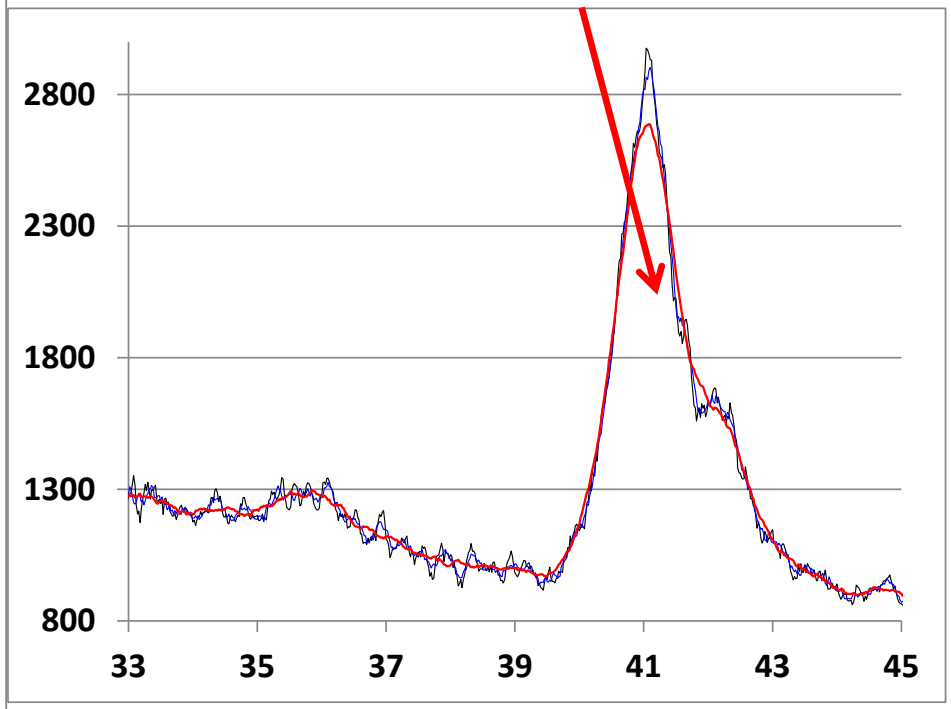
- **Moving average** (移動平均法)

  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



Simple moving average: 5,11,31点



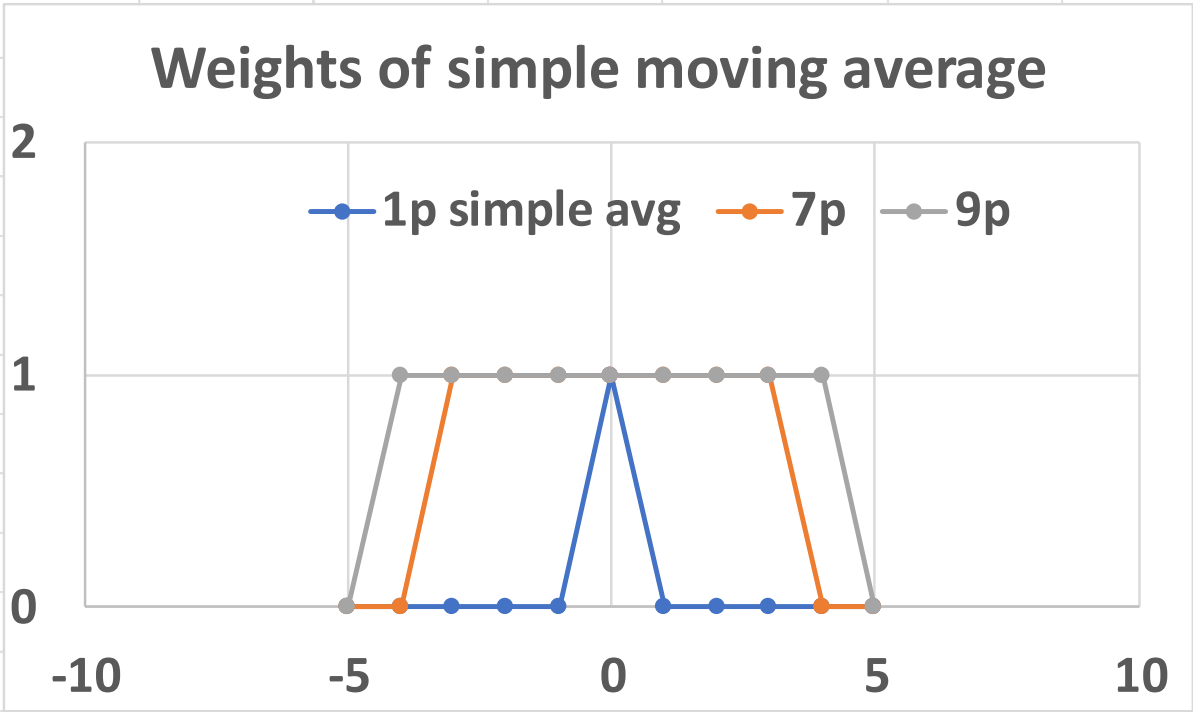
Smoothing

Simple moving average (2m+1 points)

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

Weight

i-i0	1p simple avg	7p	9p
-5	0	0	0
-4	0	0	1
-3	0	1	1
-2	0	1	1
-1	0	1	1
0	1	1	1
1	0	1	1
2	0	1	1
3	0	1	1
4	0	0	1
5	0	0	0
W	1	7	9



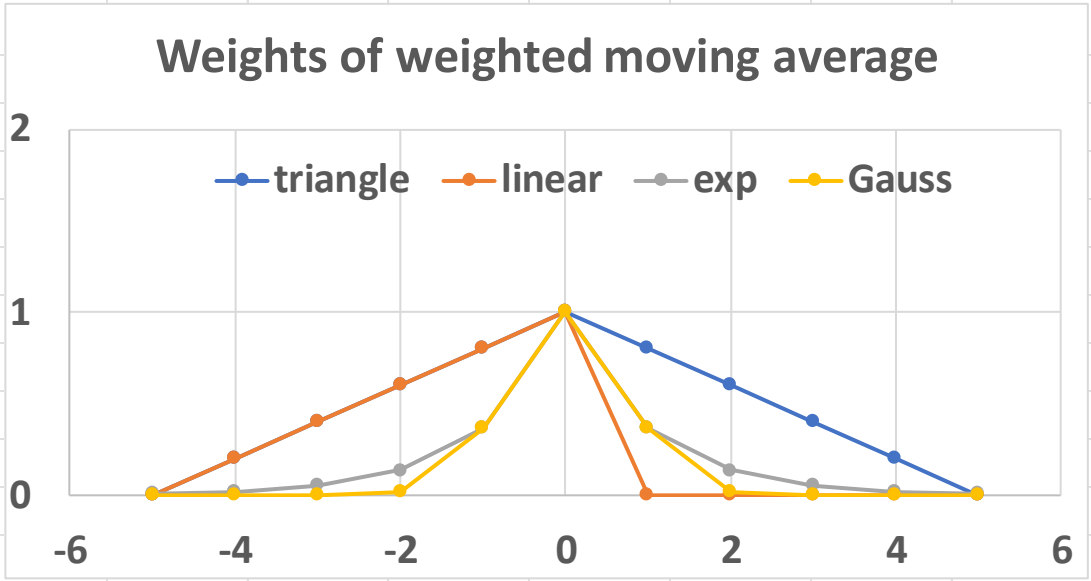
Smoothing

Weighted moving average (2m+1 points)

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

w_i

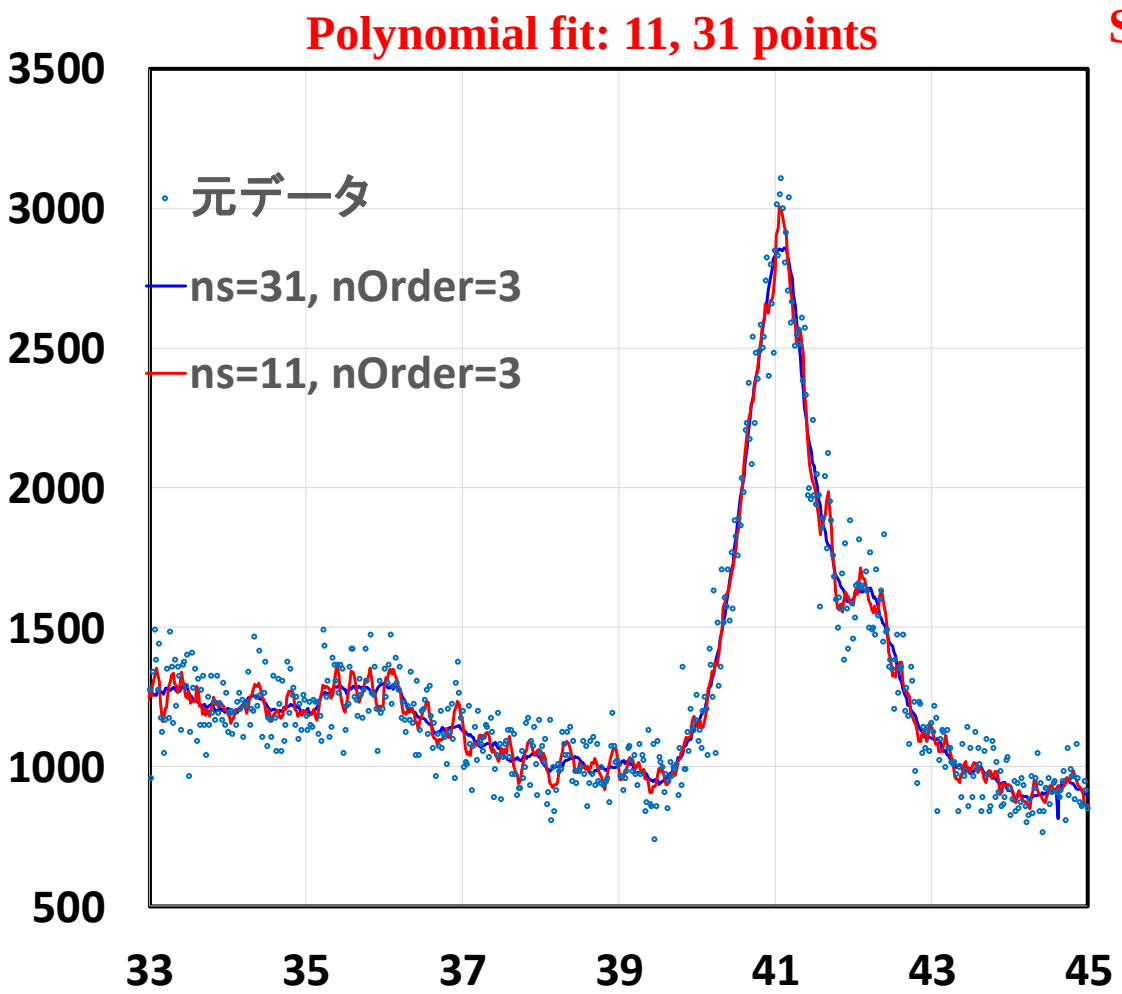
i-i0	triangle	linear	exp	Gauss
-5	0	0	0.006738	1.39E-11
-4	0.2	0.2	0.018316	1.13E-07
-3	0.4	0.4	0.049787	0.000123
-2	0.6	0.6	0.135335	0.018316
-1	0.8	0.8	0.367879	0.367879
0	1	1	1	1
1	0.8	0	0.367879	0.367879
2	0.6	0	0.135335	0.018316
3	0.4	0	0.049787	0.000123
4	0.2	0	0.018316	1.13E-07
5	0	0	0.006738	1.39E-11
W	5	3	2.156111	1.772637

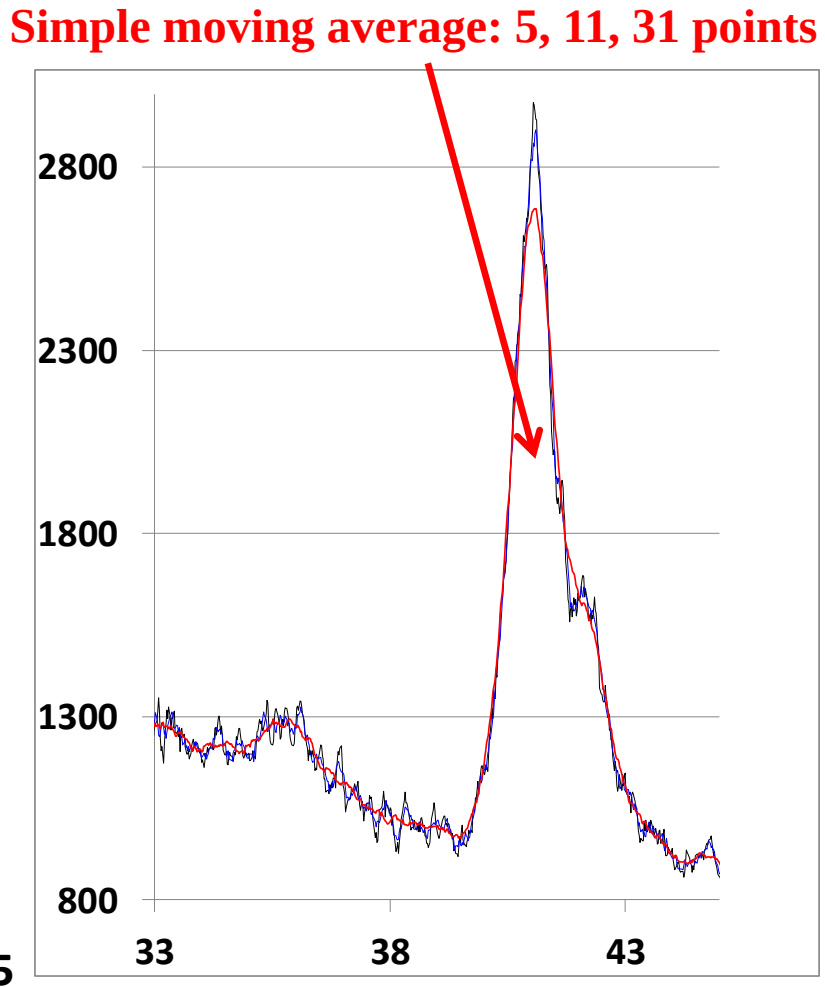


Smoothing: Polynomial fit method (多項式適合法)

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





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

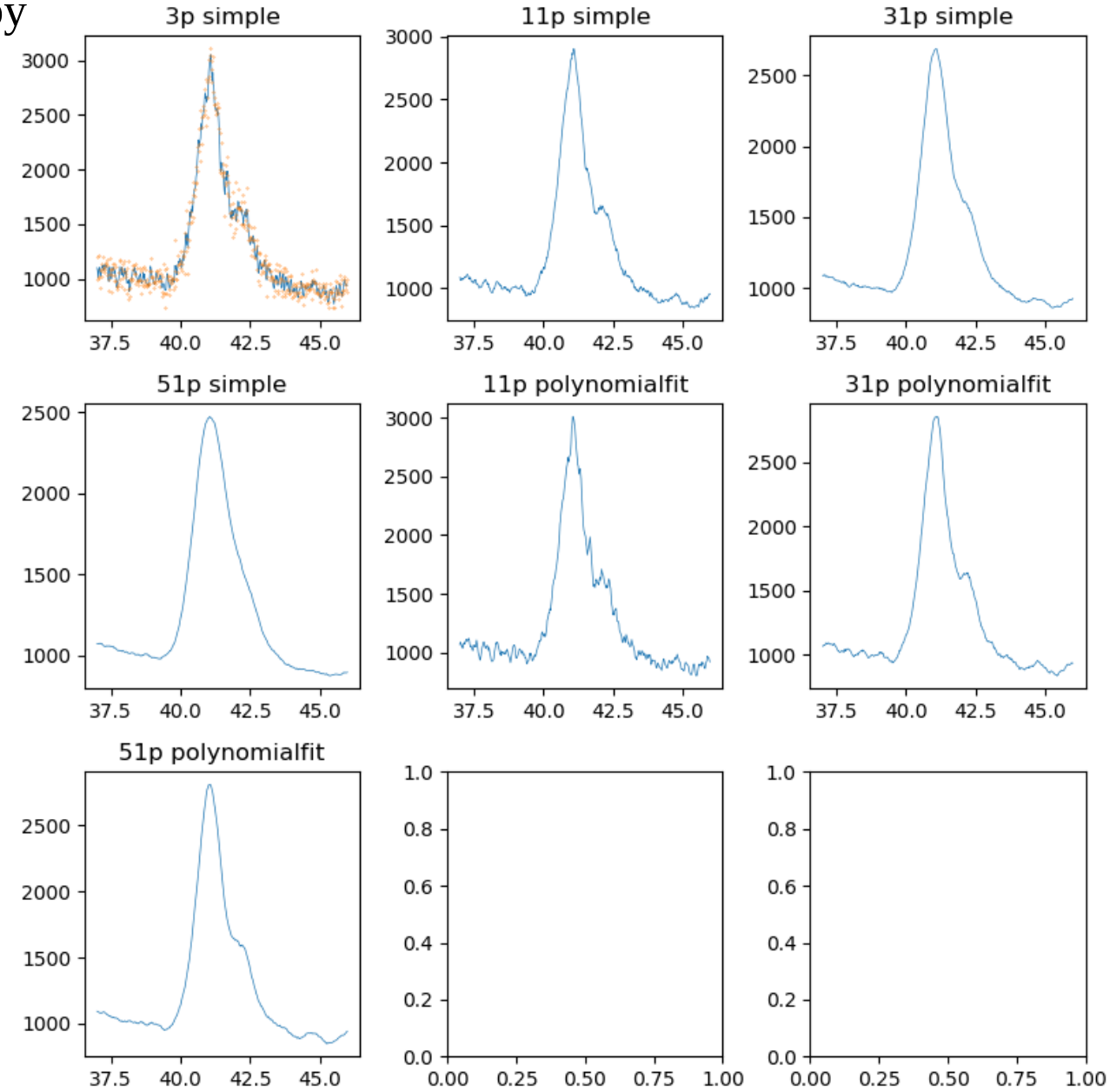
Weights for order 2 and 3 using (2*m*+1) points ((2*m*+1)点を用いた2,3次多項式適合の重み)

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

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

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

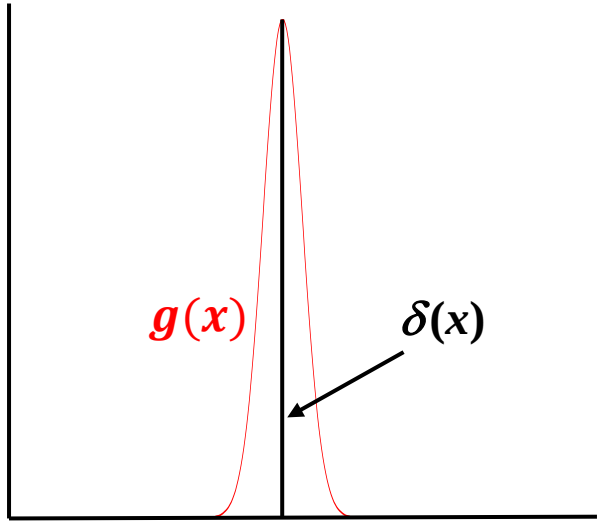
- Convert time-dependent data to frequency data
- Convert position-dependent data to wavenumber data
- Origin of original data is converted to whole range of FT data
- Whole range of original data is converted to origin of FT data
- **IFT of FTed data recovers the original data**

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)

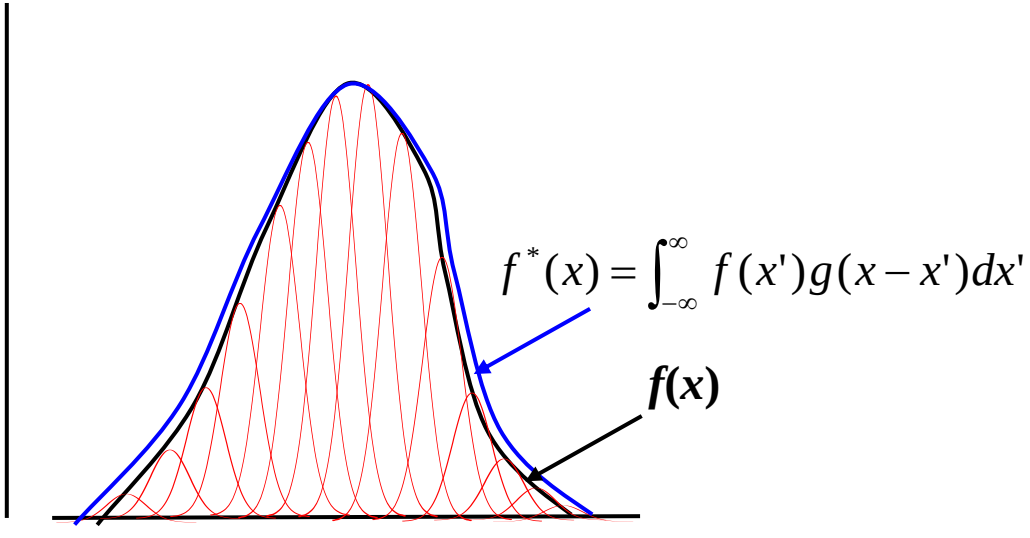
$$(f * g)(x) = f^*(x) = \int_{-\infty}^{\infty} f(x')g(x - x')dx'$$

Observed peak has a finite width originating from apparatus function $g(x)$
Even if the intrinsic peak has zero width (delta function $\delta(x)$)
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 a 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)

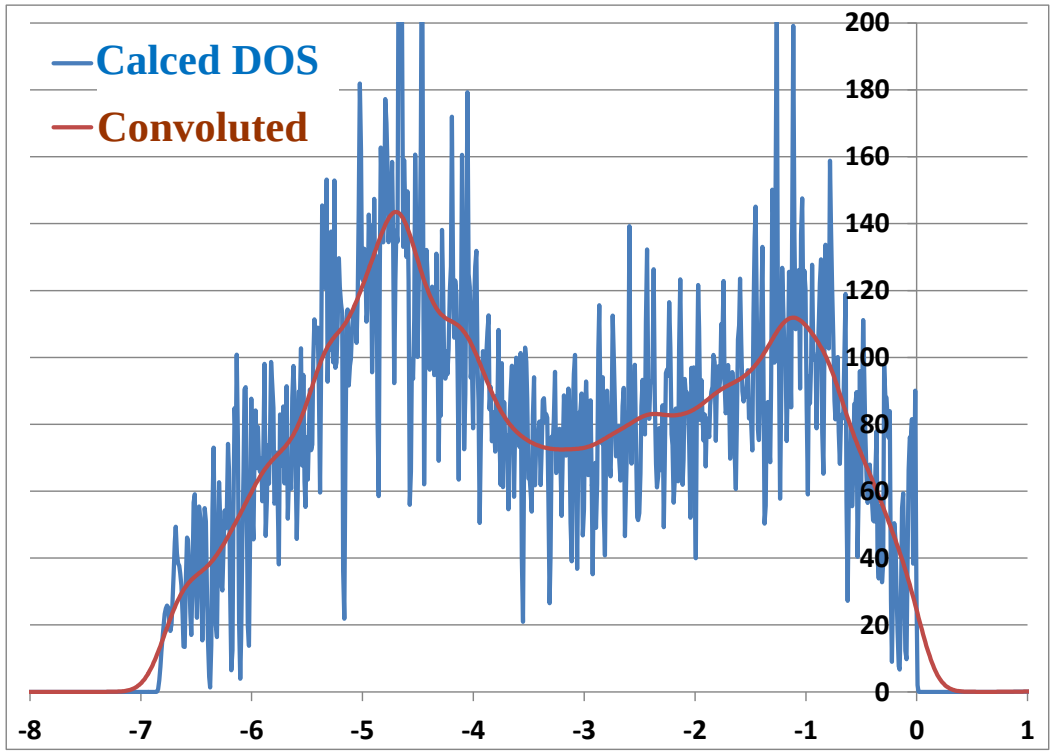
Density of state (DOS) function calculated by density functional theory

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)

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



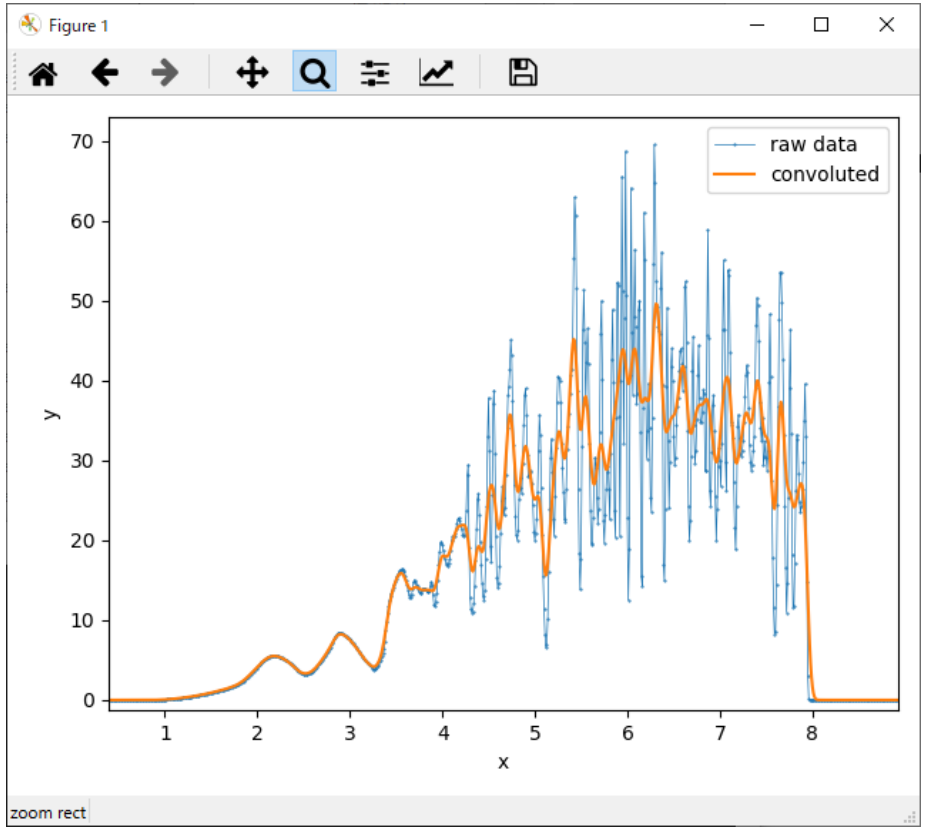
Note: Estimation of band, edge energies will have the errors originating from the smearing width *w*

Program: convolution.py

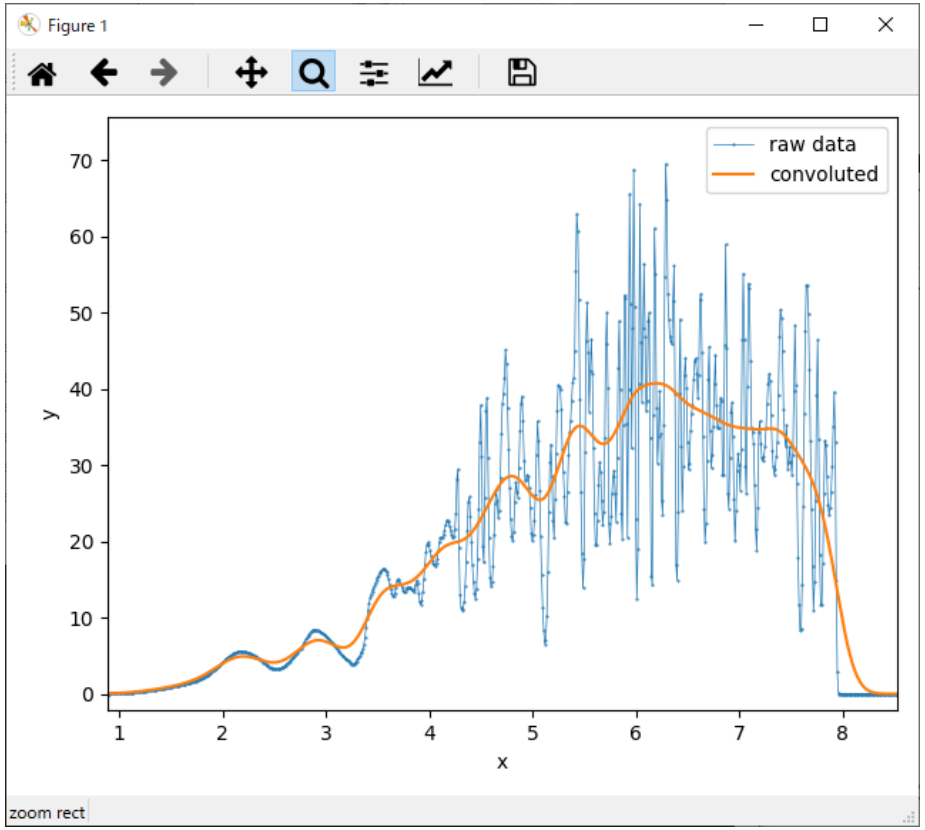
Usage: python convolution.py width

width: width of Gaussian function to convolute

python convolution.py 0.05



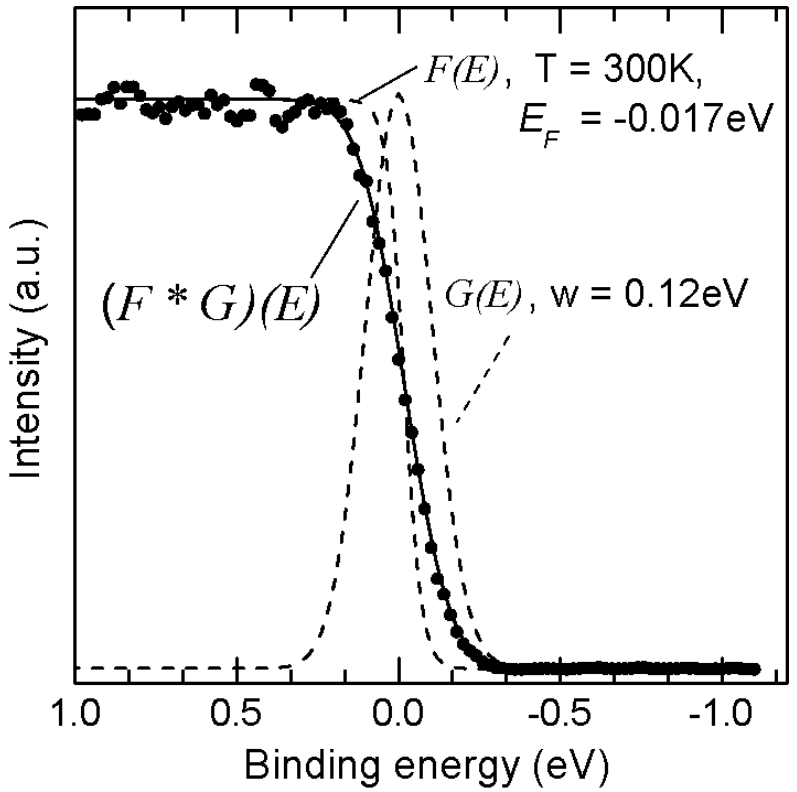
python convolution.py 0.2



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

$$\left.\frac{dy}{dx}\right|_2 = \boxed{\frac{1}{2h}(-1 \quad 0 \quad 1)}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad \left.\frac{dy}{dx}\right|_0 \sim \frac{y_1 - y_{-1}}{2h}$$

Filter

Second differential

$$\left.\frac{d^2y}{dx^2}\right|_2 = \frac{1}{2h^2}(1 \quad -2 \quad 1)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad \left.\frac{d^2y}{dx^2}\right|_0 \sim \frac{y_1 - 2y_0 + y_{-1}}{h^2}$$

Simple moving average (3 points)

$$y_{2,s} = \frac{1}{3}(1 \quad 1 \quad 1)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad y_{2,s} \sim \frac{y_1 + y_2 + y_3}{3}$$

Weighted moving average (3p one-side triangle)

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

Weighted moving average (3p double-side triangle)

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

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

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

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

Weights for various smoothing

Weighted moving average (2m+1 points)

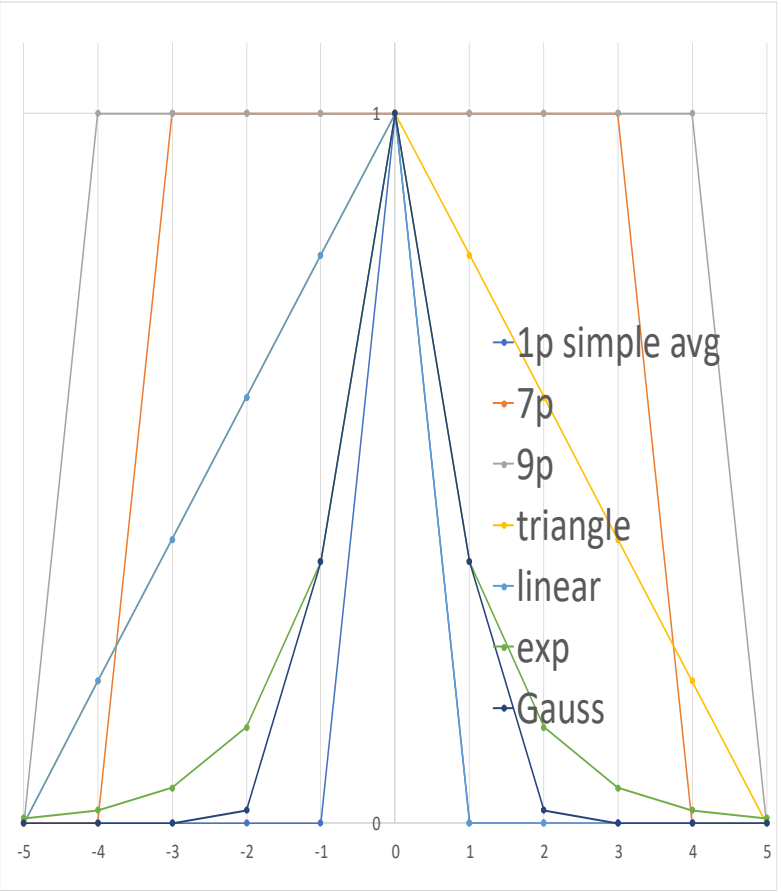
$$y_{i,smoothed} = \frac{1}{\sum_{j=i-m}^{i+m} w_i} \sum_{j=i-m}^{i+m} w_i y_j$$

e.g., one-side triangle : $(w_{-1}, w_0, w_1) = \frac{1}{3}(1,2,0)$

triangle : $(w_{-1}, w_0, w_1) = \frac{1}{4}(1,2,1)$

Gaussian : $w_i = \frac{1}{\sum e^{(ai)^2}} e^{(ai)^2}$

i-i0	1p simple avg	7p	9p	triangle	linear	exp	Gauss	3p order 3 polynomial	5p	7p
-5	0	0	0	0	0	0.006738	1.39E-11	0	0	0
-4	0	0	1	0.2	0.2	0.018316	1.13E-07	0	0	0
-3	0	1	1	0.4	0.4	0.049787	0.000123	0	0	-10
-2	0	1	1	0.6	0.6	0.135335	0.018316	0	-3	15
-1	0	1	1	0.8	0.8	0.367879	0.367879	0	12	30
0	1	1	1	1	1	1	1	5	17	35
1	0	1	1	0.8	0	0.367879	0.367879	0	12	30
2	0	1	1	0.6	0	0.135335	0.018316	0	-3	15
3	0	1	1	0.4	0	0.049787	0.000123	0	0	-10
4	0	0	1	0.2	0	0.018316	1.13E-07	0	0	0
5	0	0	0	0	0	0.006738	1.39E-11	0	0	0
W	1	7	9	5	3	2.15611	1.77264	5	35	105
							m=	1	2	3



Equivalent to convolution using these functions

Weights of polynomial smoothing

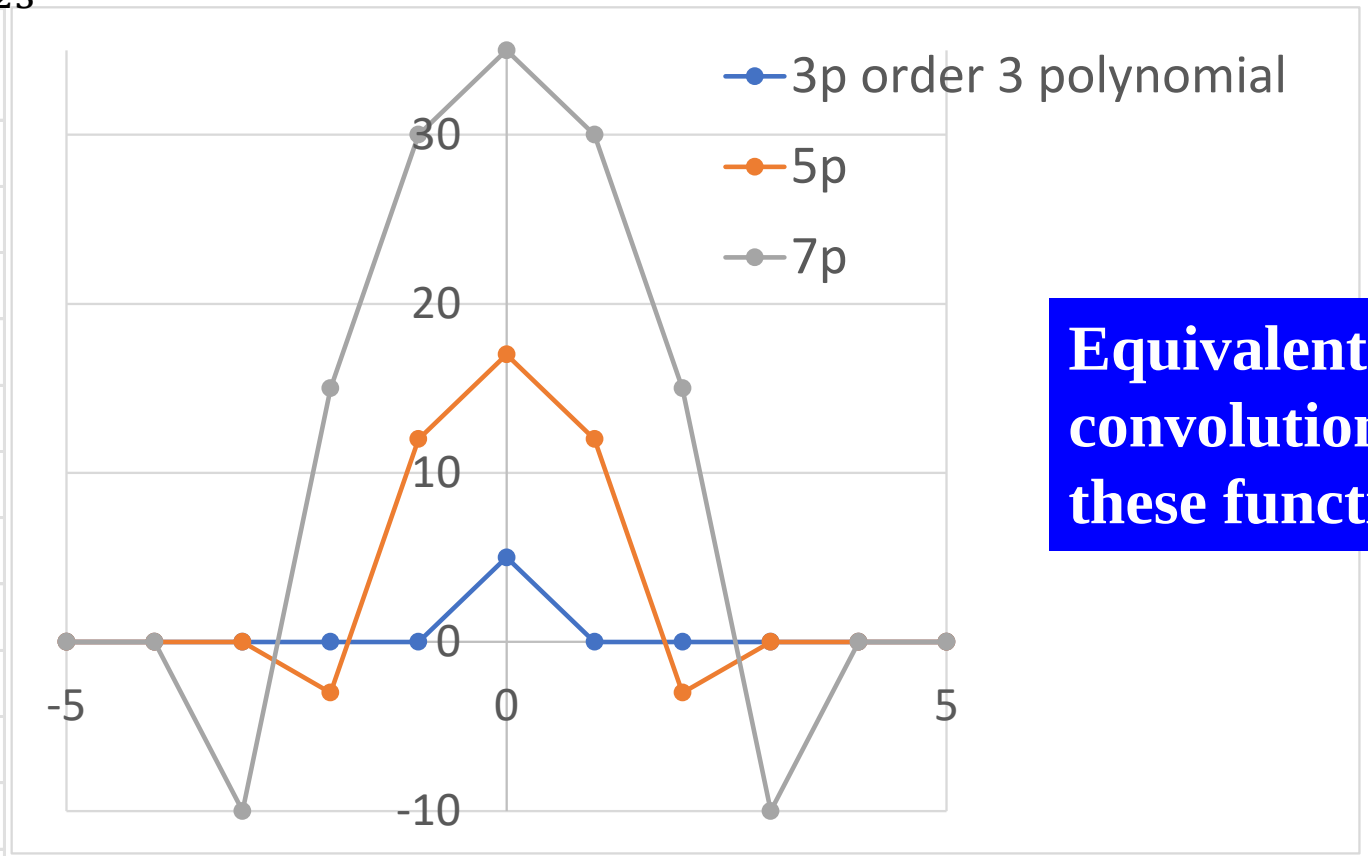
Order 2 and 3 polynomial fit using (2*m*+1) points

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

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

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

i-i0	3p order 3 polynomial	5p	7p
-5	0	0	0
-4	0	0	0
-3	0	0	-10
-2	0	-3	15
-1	0	12	30
0	5	17	35
1	0	12	30
2	0	-3	15
3	0	0	-10
4	0	0	0
5	0	0	0
W	**5**	**35**	**105**
	1	2	3



Equivalent to convolution using these functions

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

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)

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

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

$$F^*(k) = \int_{-\infty}^{\infty} f(x')g(x-x')\exp(ikx)dxdx'$$

$$= \int_{-\infty}^{\infty} f(x')\left(\int g(x-x')\exp(ikx)dx\right)dx'$$

$$= \int_{-\infty}^{\infty} f(x')\left(\int g(x)\exp(ik(x+x'))dx\right)dx'$$

$$= \int_{-\infty}^{\infty} f(x')G(k)\exp(ikx')dx'$$

$$= F(k)G(k)$$

$f(x)$ 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, $x_i^{(k+1)}$ is estimated from $x_i^{(k)}$

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

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

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

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

Using the known $x_j^{(k+1)}$ 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.

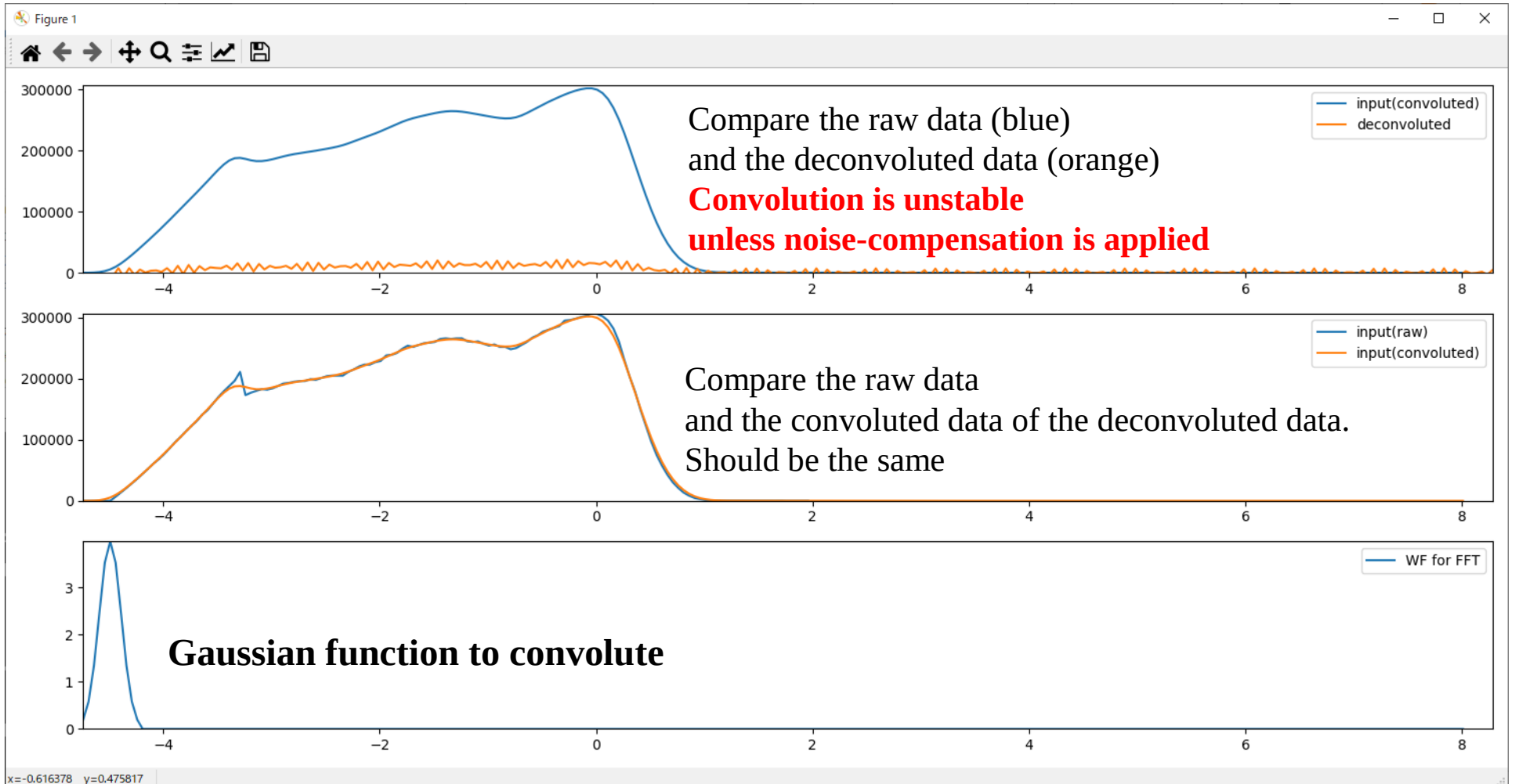
Program: deconvolution.py (mode=fft)

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**



Program: Gauss-Seidel method with smoothing

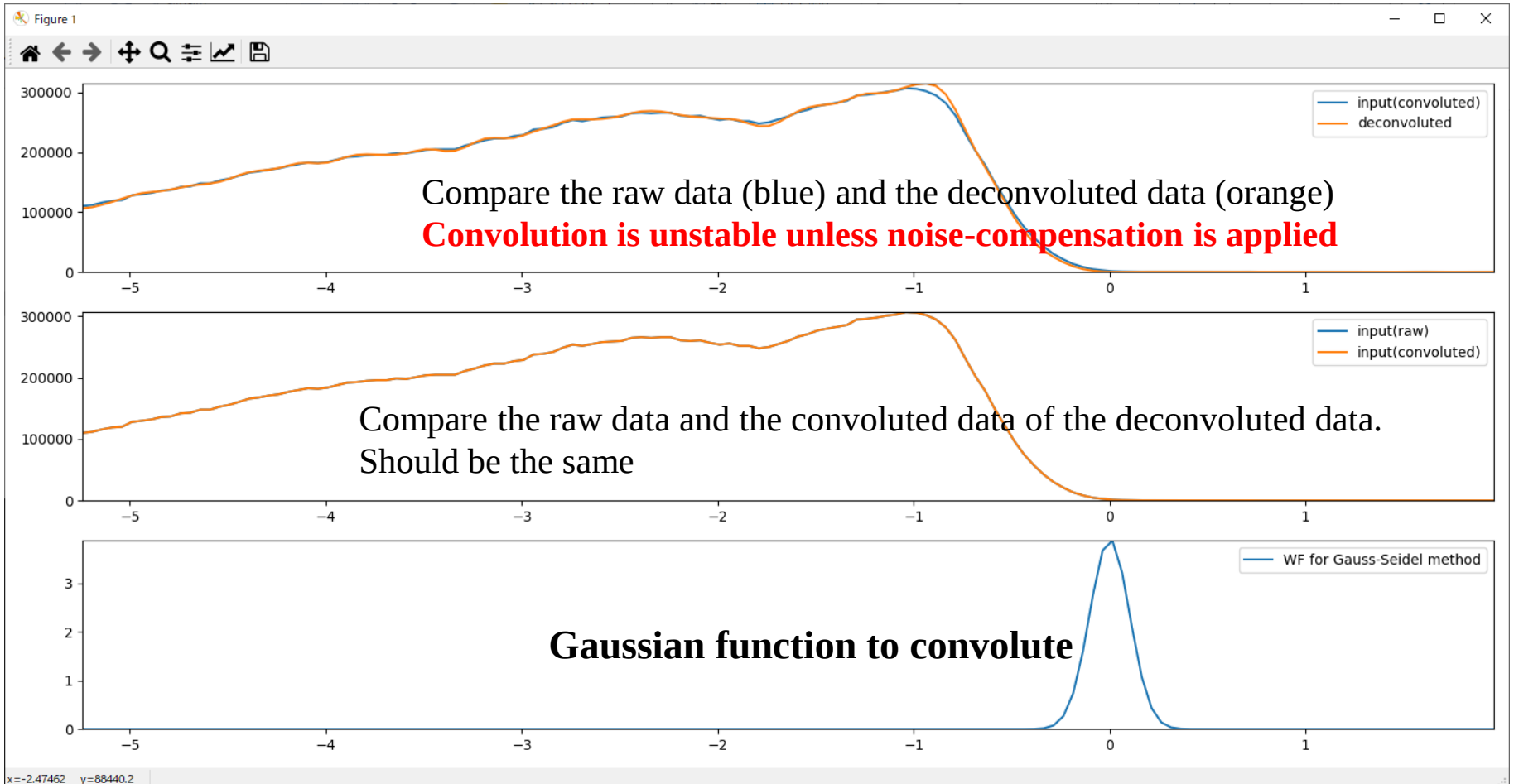
Usage: python deconvolution.py file mode xmin xmax Wa dump nmaxiter eps nsmooth zeroc

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.



Deconvolution: Gauss-Seidel method w/o smooting

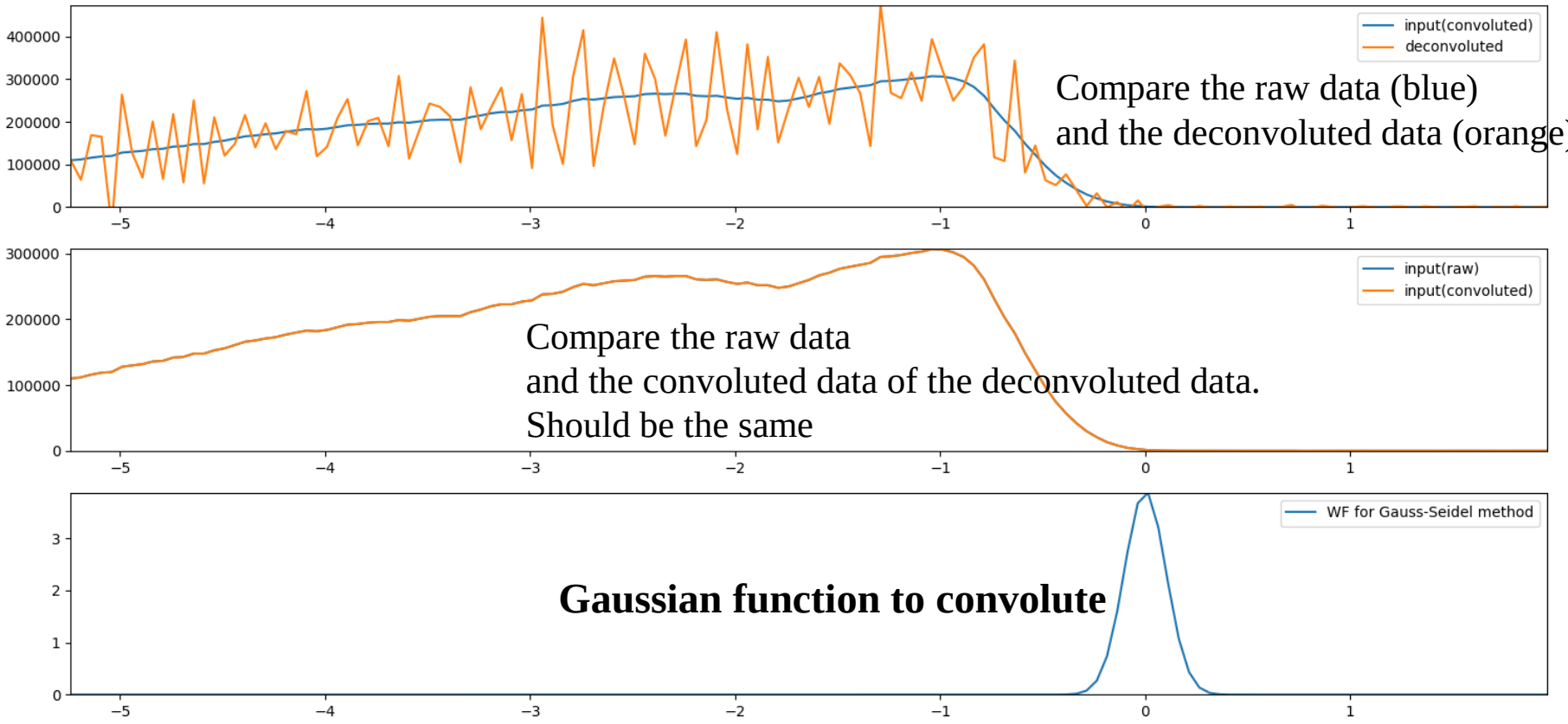
Usage: python deconvolution.py file mode xmin xmax Wa dump nmaxiter eps nsmooth zeroc

see usage of the program output

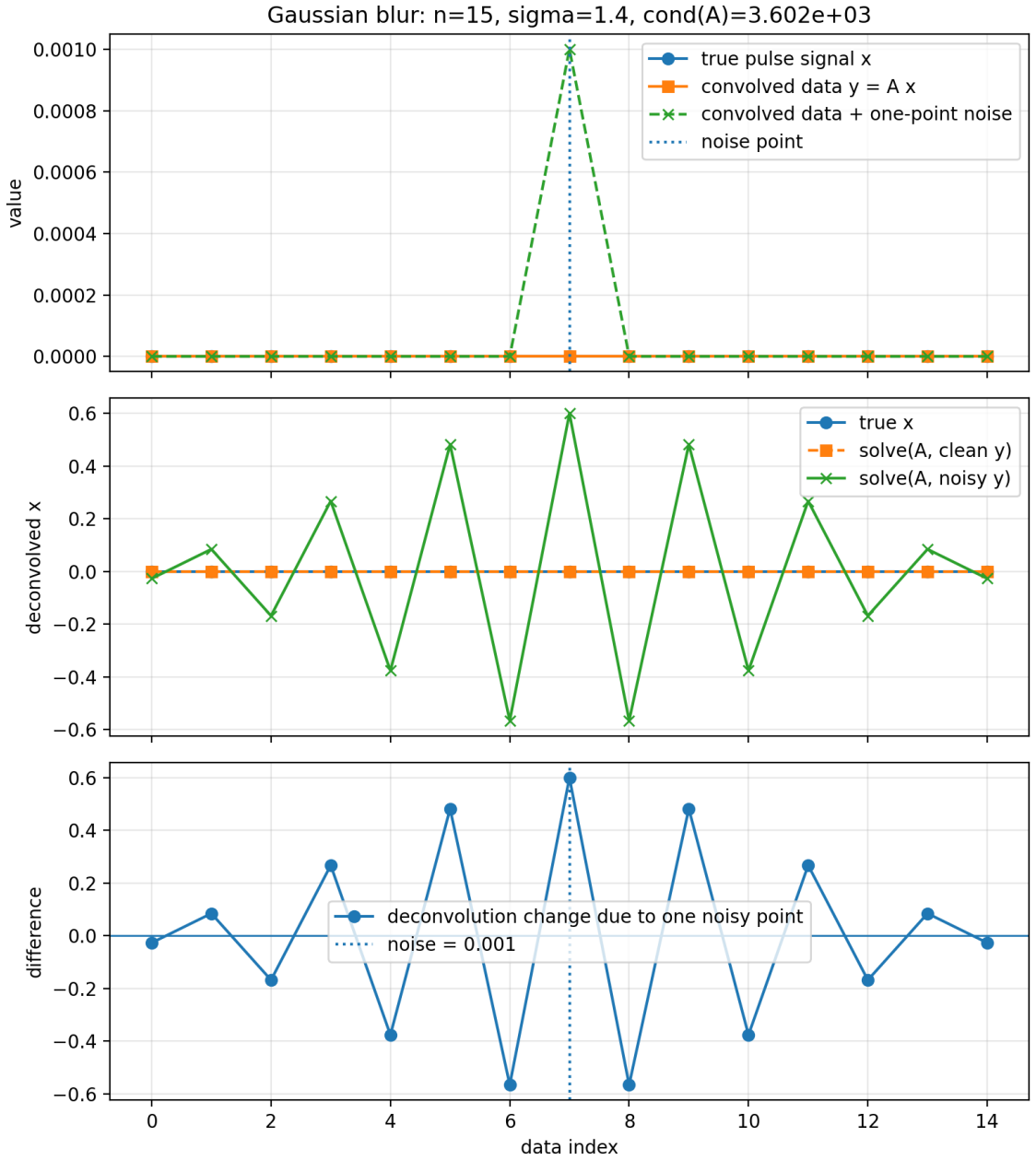
python deconvolution.py pes.csv gs -6.0 2.0 0.12 1.0 300 1.0e-4 1 0

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

No smoothing is applied for each iteration.

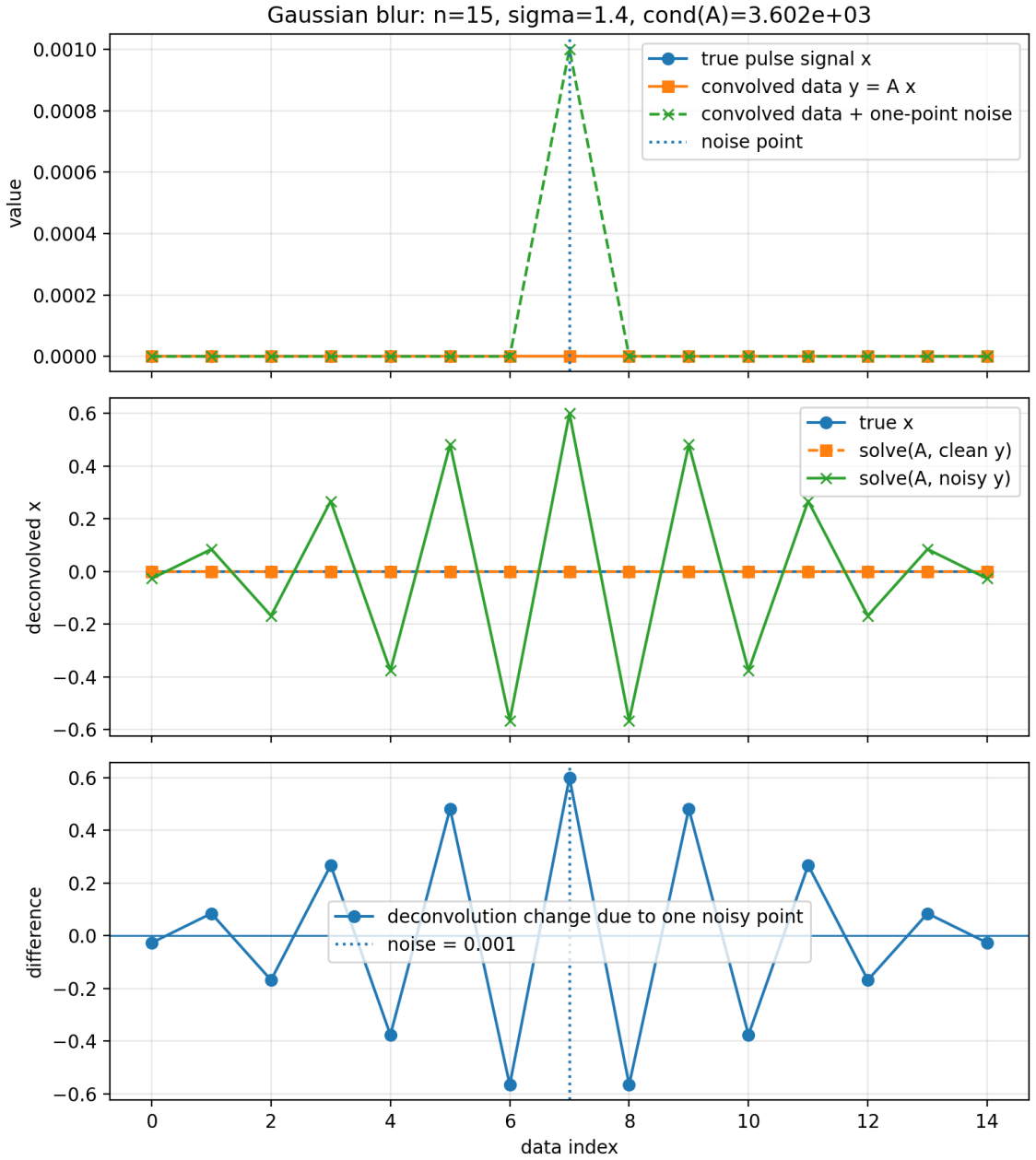


Why is deconvolution sensitive to noise?: deconv_noise_demo.py



- If no true signal exists, the ideal measured data should be zero everywhere.
- However, if even a small error is introduced at one point in the measured data, direct deconvolution tries to explain that error as part of the true spectral structure.
- In this situation, the direct method cannot treat the error at only that single point.
- In convolution with a finite width, each measured point is produced from signals at multiple neighboring points. Therefore, neighboring points are coupled in the system of linear equations.
- As a result, a correction at one point also affects the equation for the neighboring point, and the effect propagates successively to the next neighboring points.
- Consequently, false structures that oscillate between positive and negative values spread throughout the reconstructed spectrum.

なぜデコンボリューションはノイズに弱いのか: deconv_noise_demo.py



- 真の信号が存在しない場合、理想的な測定データはすべてゼロになる。
- しかし、測定データに誤差が入ると、直接解法によるデコンボリューションは、その誤差を真のスペクトル構造として説明しようとする。
- このとき、直接解法は一点の誤差をその点だけで処理できない。
- 幅を持つ畳み込みでは、各測定点が周囲の複数点の信号から作られるため、隣接点同士が連立方程式の中で結合している。
- そのため、ある点の補正は隣の点の方程式にも影響し、さらにその隣へと連鎖的に伝搬する。
- その結果、復元スペクトルには正負に振動する偽の構造が広がる。