材料計算科学・データ解析に関するチュートリアルコース
2023/3/22 10:00~11:00

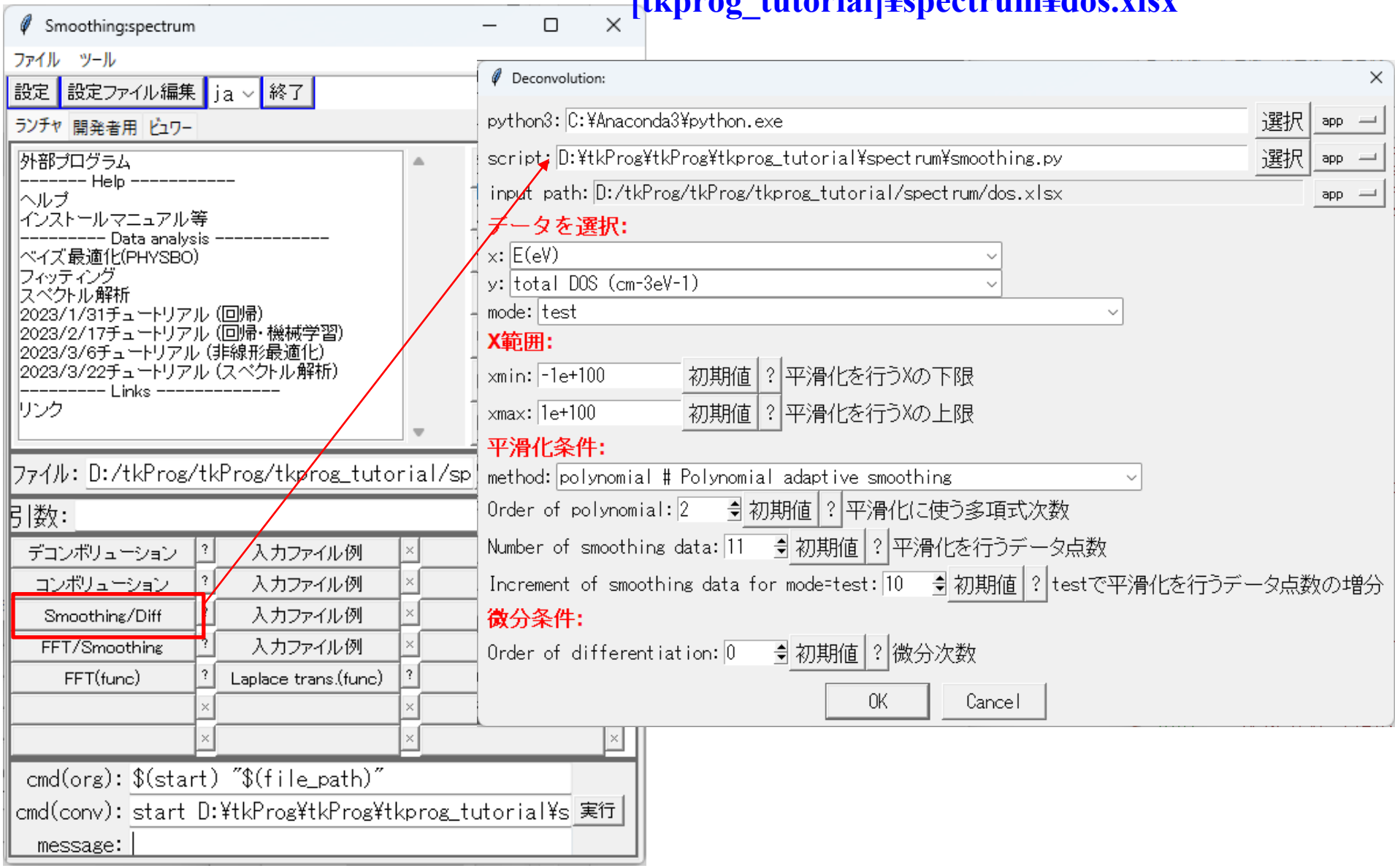
スペクトル解析

Smoothing

平滑化

平滑化

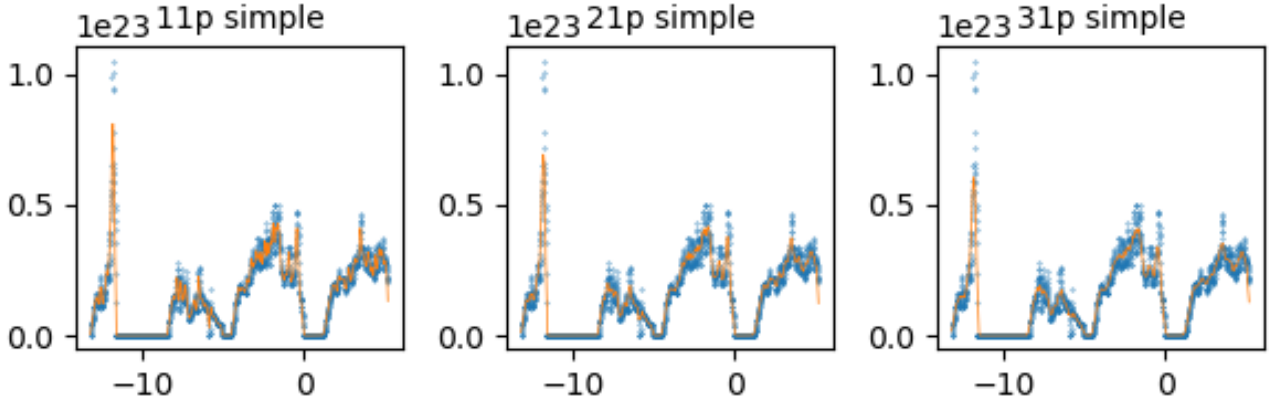
[tkprog_tutorial]¥spectrum¥dos.xlsx



平滑化の結果

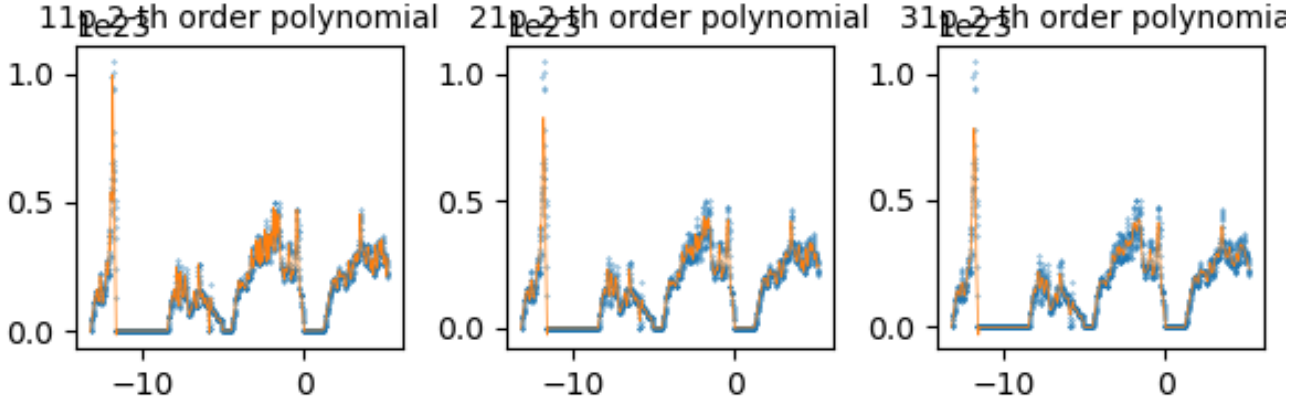
単純移動平均

平滑化点数を11,21,31点



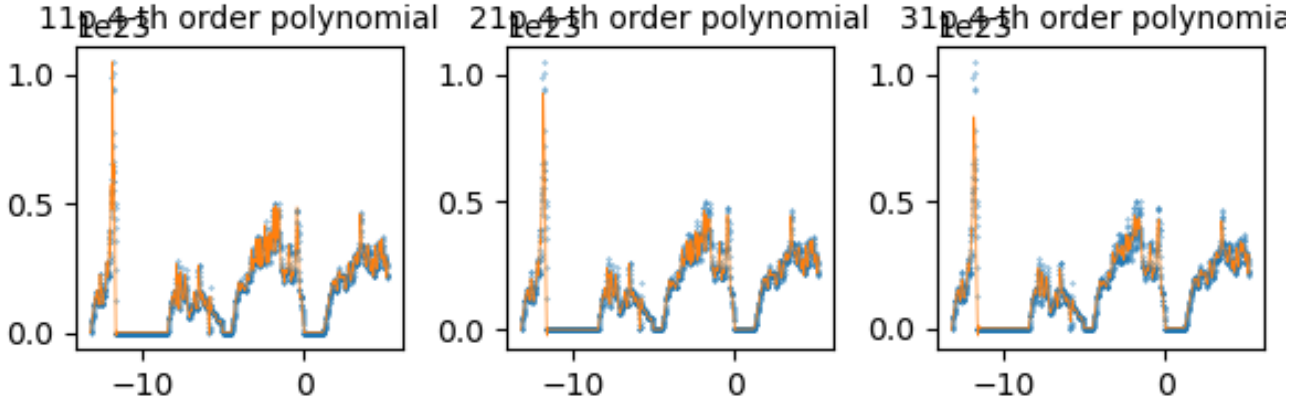
2次多項式平滑化

平滑化点数を11,21,31点



4次多項式平滑化

平滑化点数を11,21,31点



平滑化

着目している点 N点 のある平均を取る

- 単純移動平均
  重みを均一に 1/N とする
- 加重移動平均
  重み: 線形、三角形、指数関数、Gauss関数 など

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)

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

何らかの関数で近似して平滑化を行う

- **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)

その他

- **Fourier transformation** (フーリエ変換)

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)

2次および3次多項式適合法の重み

# 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**

1次: 単純移動平均
2次と3次は同じ重みになる

(2m+1)点を用いた2,3次多項式適合の重み

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

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

高次の微分公式

3点公式

$$f'(a) = \frac{1}{h}\left\{\frac{1}{2}f(a+h) - \frac{1}{2}f(a-h)\right\} + \frac{1}{6}f^{(3)}(a)h^2 + \cdots$$

5点公式

$$f'(a) = \frac{1}{h}\left\{-\frac{1}{12}f(a+2h) + \frac{2}{3}f(a+h) - \frac{2}{3}f(a-h) + \frac{1}{12}f(a-2h)\right\} + \frac{1}{30}f^{(5)}(a)h^4 + \cdots$$

7点公式

$$f'(a) = \frac{1}{h}\left\{\frac{1}{60}f(a+3h) - \frac{3}{20}f(a+2h) + \frac{3}{4}f(a+h) - \frac{3}{4}f(a-h) + \frac{3}{20}f(a-2h) - \frac{1}{60}f(a-3h)\right\} + \frac{1}{140}f^{(7)}(a)h^6 + \cdots$$

フィルターとコンボリューション

コンボリューション:
データとフィルターの
行列演算

中央差分1次微分

$$\frac{dy}{dx_2} = \boxed{\frac{1}{2h}(-1 \quad 0 \quad 1)\ \text{フィルター}} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

中央差分2次微分

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

単純移動平均 (3点)

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

加重移動平均 (3点片三角重み)

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

加重移動平均 (3点両三角重み)

$$y_{2,s} = \frac{1}{6}(1 \quad 2 \quad 1) \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

2次および3次多項式適合化平均 ($2m$+1 点)

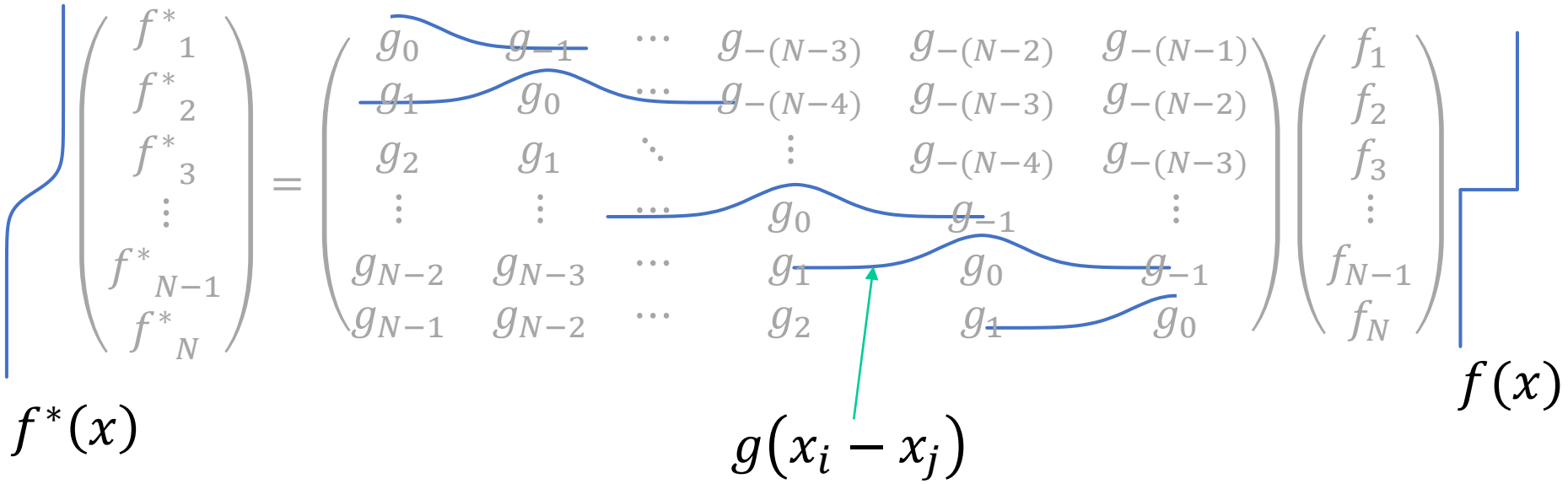
$$y_{3,s} = \frac{1}{35}(-3 \quad 12 \quad 17 \quad 12 \quad -3) \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

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

任意関数のコンボリューション

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)

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



pythonライブラリィによる平滑化

Smoothing.py

単純移動平均

```
# 1/N の重みのフィルターを作る
    w = np.ones(nsmooth) / nsmooth
# コンボリューション
    ys = np.convolve(y, w, mode = ‘same‘)
```

多項式適合化平滑化: **Savizky-Golayフィルター**

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

nsmooth: 平滑化点数

norder: 多項式の次数

deriv: 微分次数

注意: savgol_filter() では deriv = 1とすると1次微分を取ってくれるが、
x軸のデータを与えていないので、絶対値は異なる。
絶対値が必要な場合、平滑化した後、h^{deriv} で割ること

注: savgol_diff_test.py で限定的に確認した範囲です

多次元フィルター、コンボリューションと画像解析

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

フィルタ$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$の

各要素の積の和を

y_{22}のコンボリューションにとる

数学のデコンボリューションとは異なるので注意

X軸微分フィルター (エッジ検出)　$\frac{1}{2h}\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$

Y軸微分フィルター (エッジ検出)　$\frac{1}{2h}\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

斜め微分フィルター (エッジ検出)　$\frac{1}{2h}\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$

平滑化フィルター (ぼかし)　⇔ デコンボリューションにより sharpeningができる

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

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

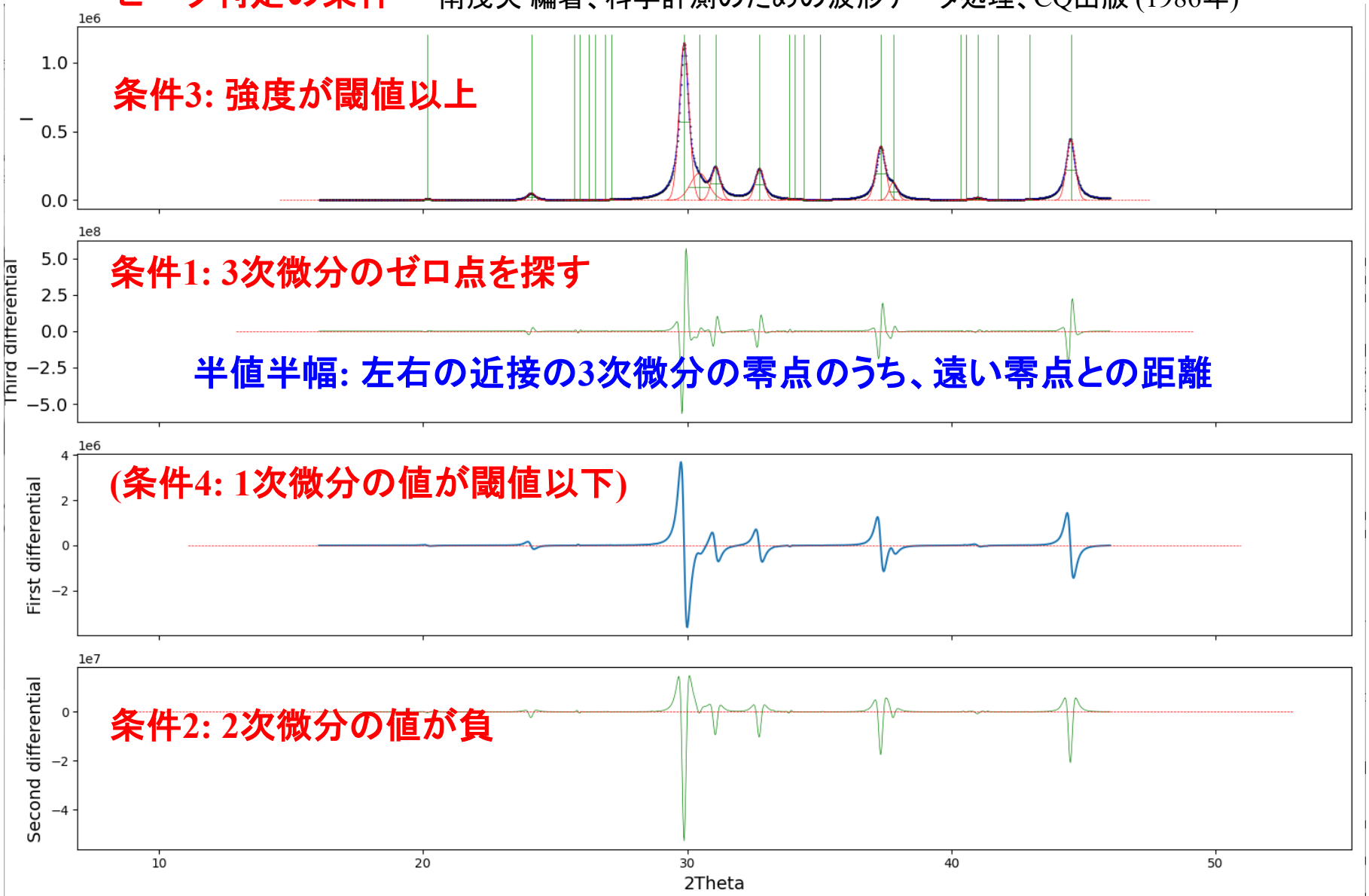
Neural network の中段にフィルターによる畳み込み層を入れて、
フィルターの要素の値を学習させる

学習したフィルターを見ることで、どのような処理が必要か理解できることもある

Peak search

[tkProg]¥tkprog_tutorial¥spectrum¥peak_search. py

ピーク判定の条件　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)



フーリエ変換

異なる定義があるので注意

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)　$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\exp(-i\omega t)d\omega$

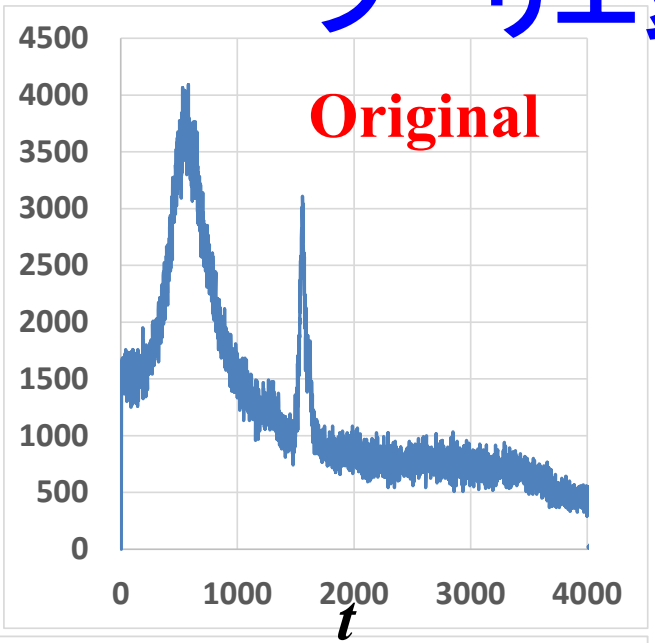
FT　$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(i2\pi ft)dt$

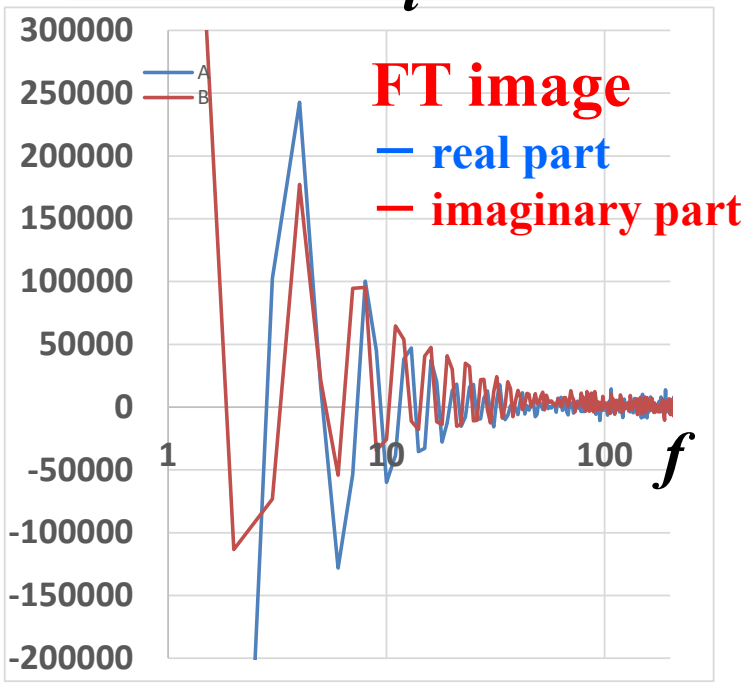
IFT　$f(t) = \int_{-\infty}^{\infty} F(\omega)\exp(-i2\pi ft)d\omega$

フーリエ変換の特徴

- 時間依存データを周波数依存データに変換
- 位置依存データを波数依存データに変換
- 元データの原点は逆空間全体に広がる
- 元データの全体は逆空間の原点に収束する
- フーリエ変換したデータ (FT) を逆変換 (IFT) すると元のデータに戻る

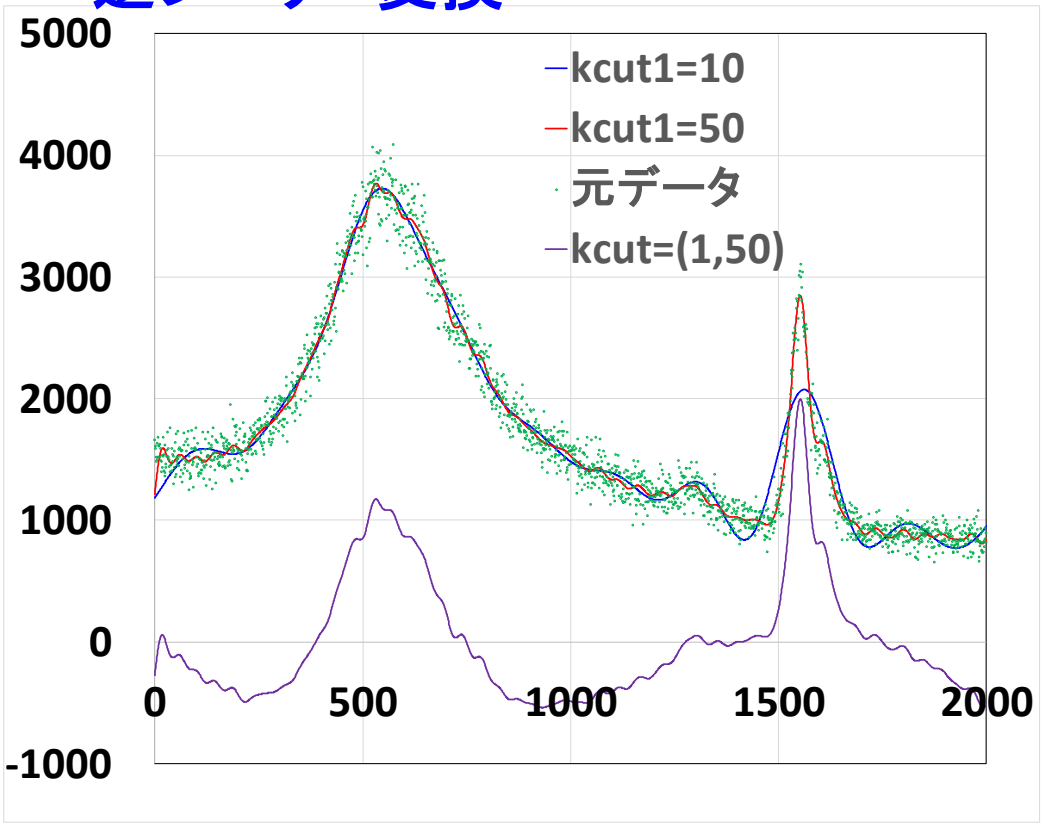
フーリエ変換による平滑化





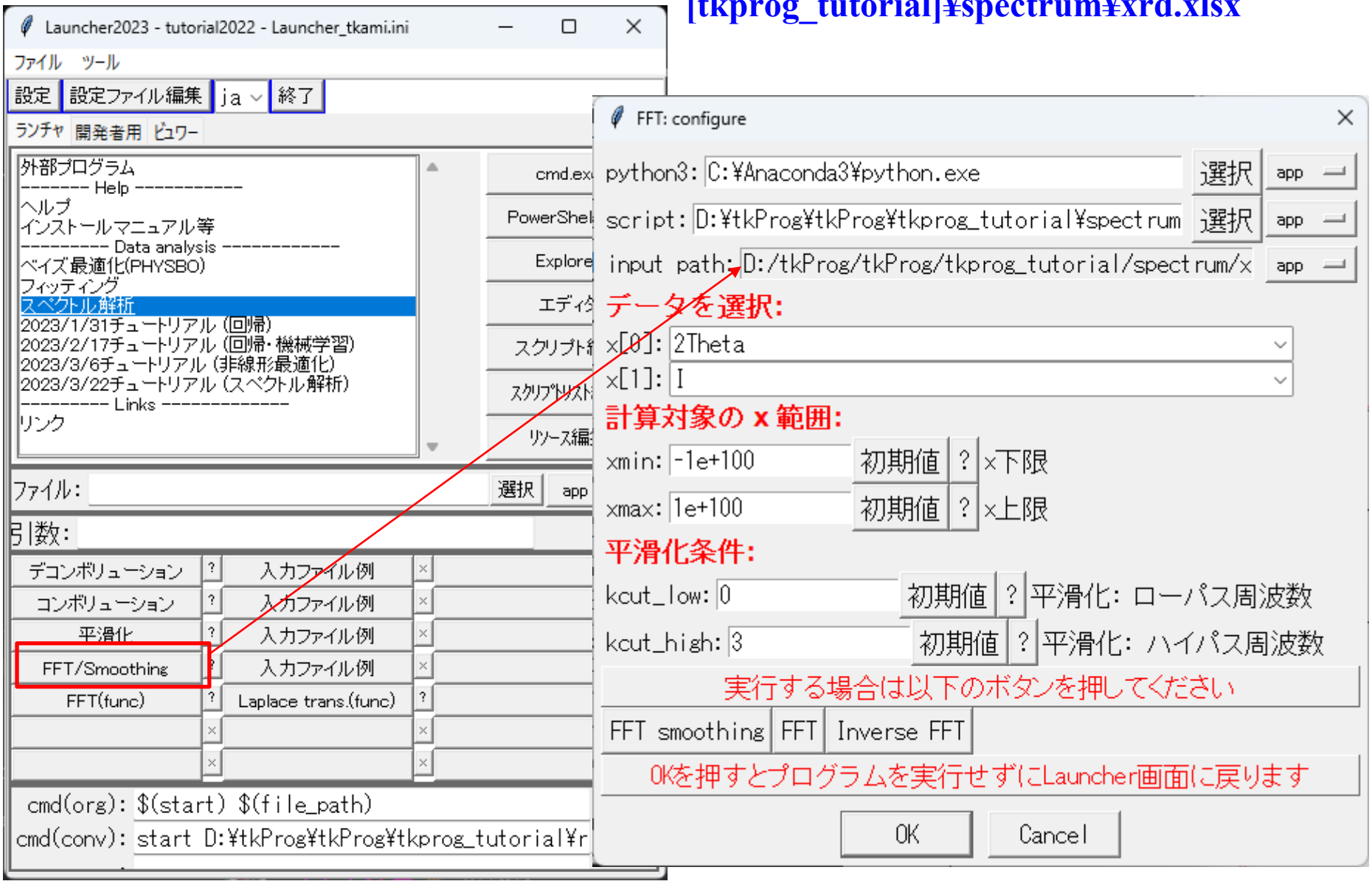
FTデータの高周波成分を除去: 平滑化
ローパスフィルター
低周波成分を除去: ドリフト成分を除去
ハイパスフィルター

例: $[k_{cut0}, k_{cut1}]$外部のデータを除去して
逆フーリエ変換



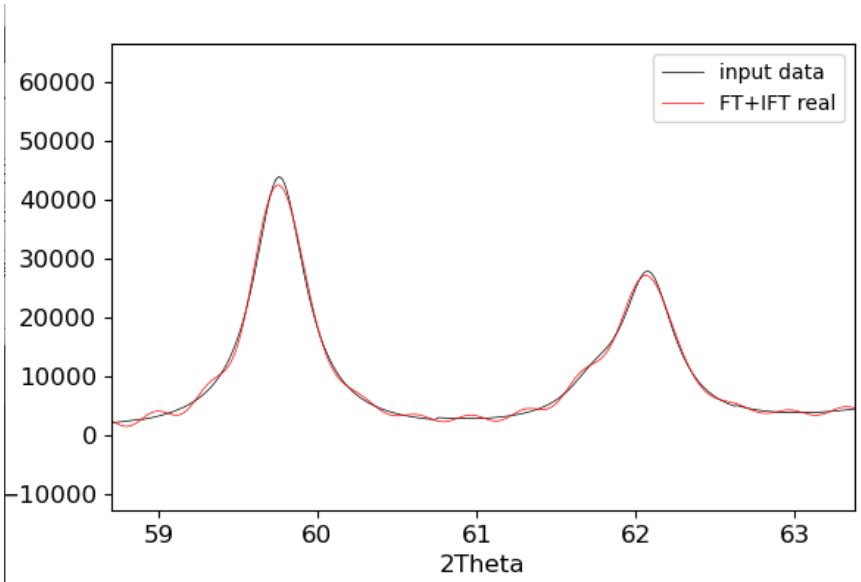
Program: smoothing-fft.py

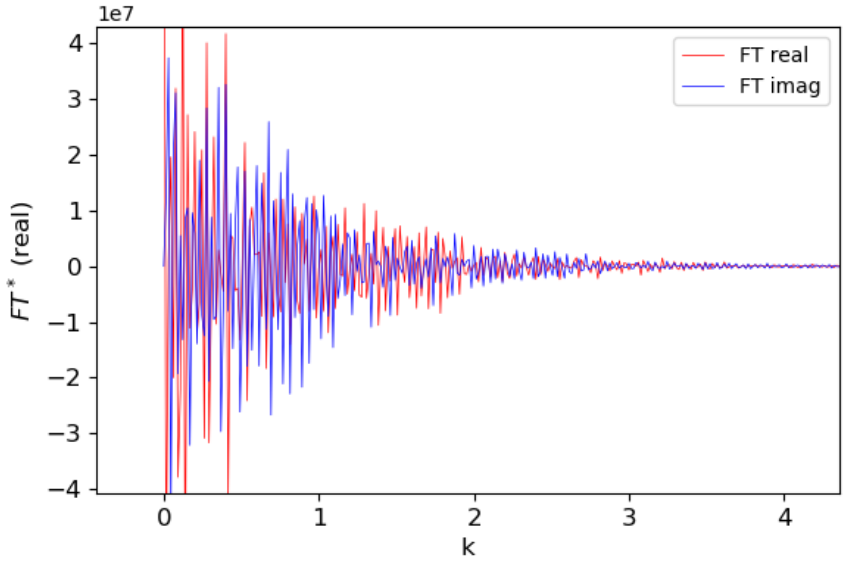
[tkprog_tutorial]¥spectrum¥xrd.xlsx

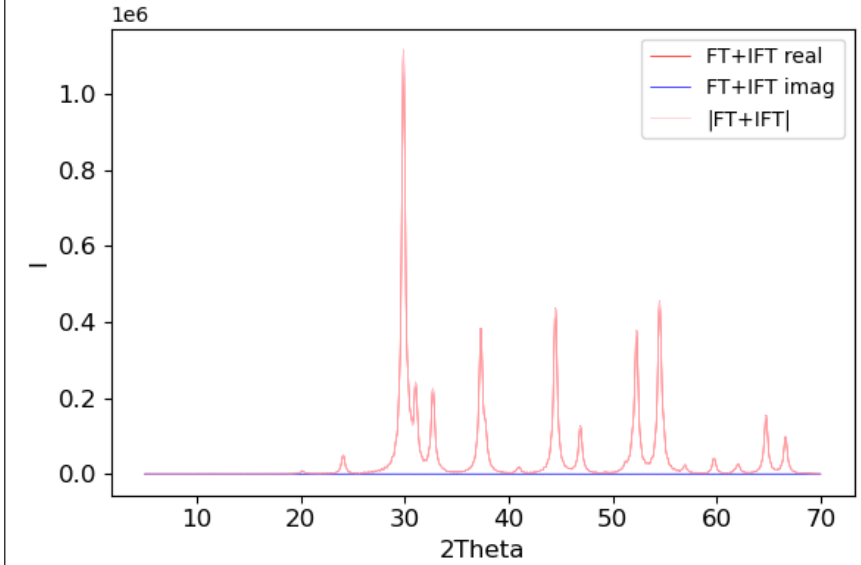


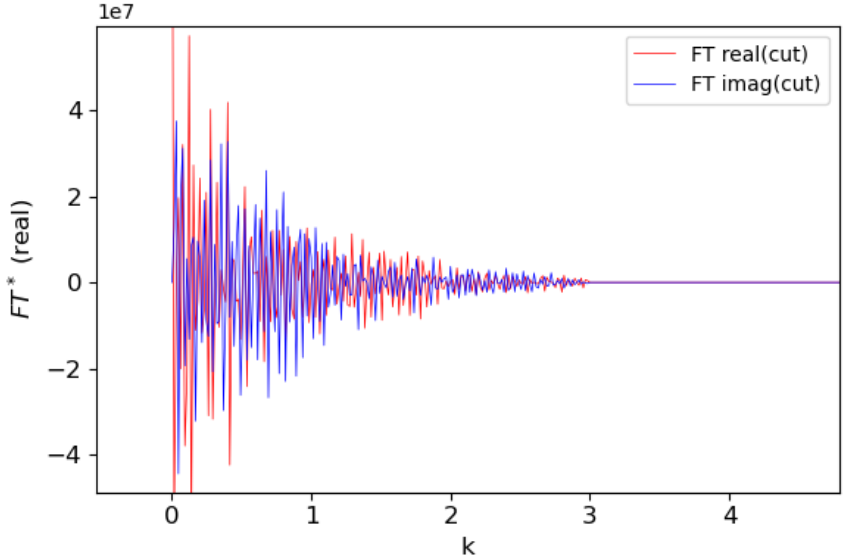
Program: smoothing-fft.py

kcut0: 0, kcut1: 3





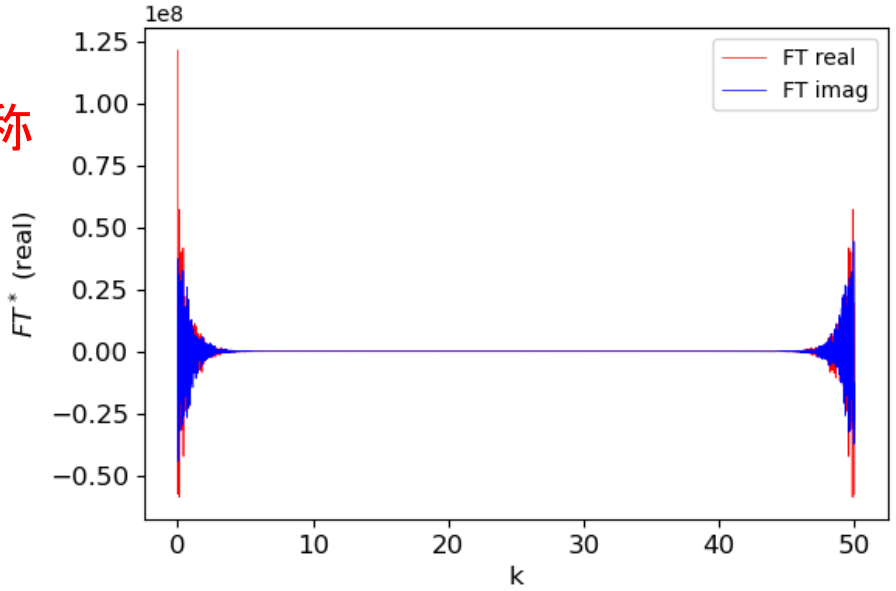




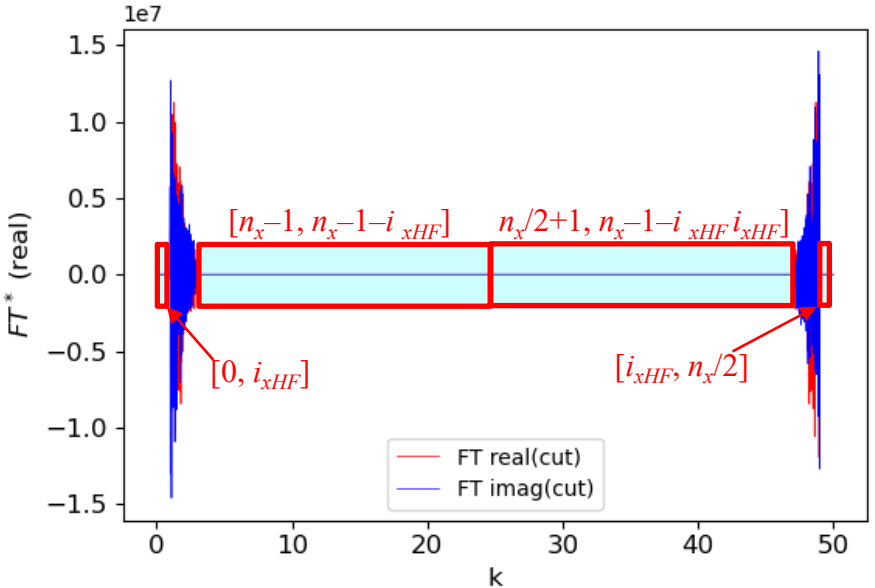
FFTの注意

Numpy fft module:　F = np.fft.fft(y)　FFT

フーリエ変換したデータは
逆座標 k の中点 ($i_x = n_x/2$) に対して鏡面対称
(フーリエ像の周期性と、実部が対称、
虚部が反対称関数であることによる).



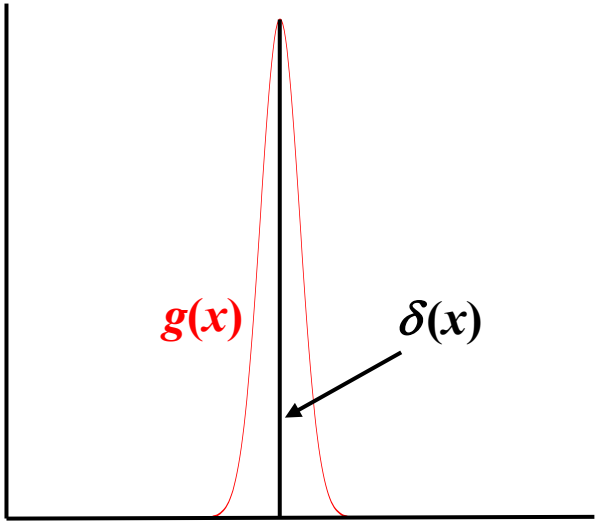
平滑化のためには
$i_x = [0, i_{xHF}]$, $[i_{xHF}, n_x/2]$ 、$[n_x-1, n_x-1-i_{xHF}]$, $n_x/2+1, n_x-1-i_{xHF}\, i_{xHF}]$の領域のデータを
ゼロにしてから逆変換を行う



畳み込み (Convolution)

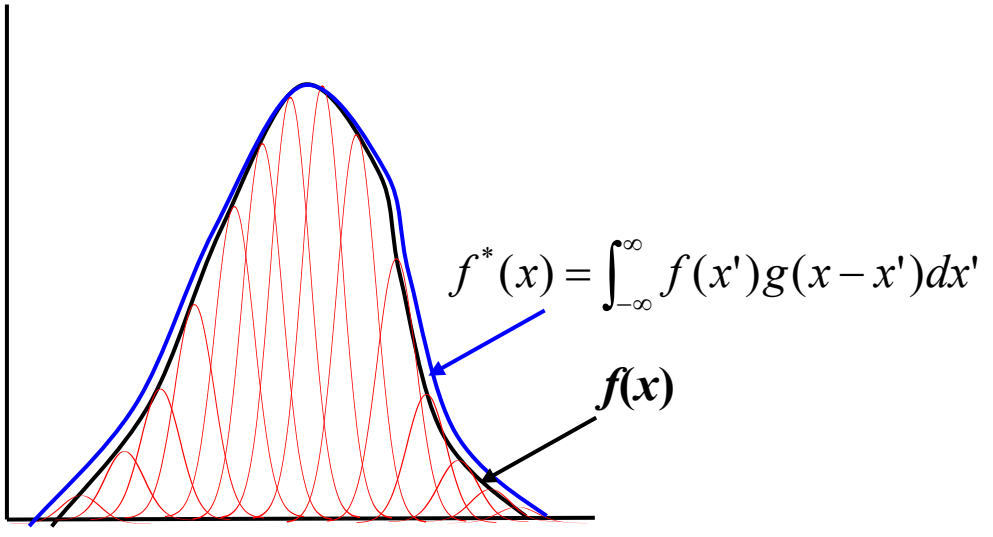
$$(f * g)(x) = f^*(x) = \int_{-\infty}^{\infty} f(x')g(x-x')dx'$$

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



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

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

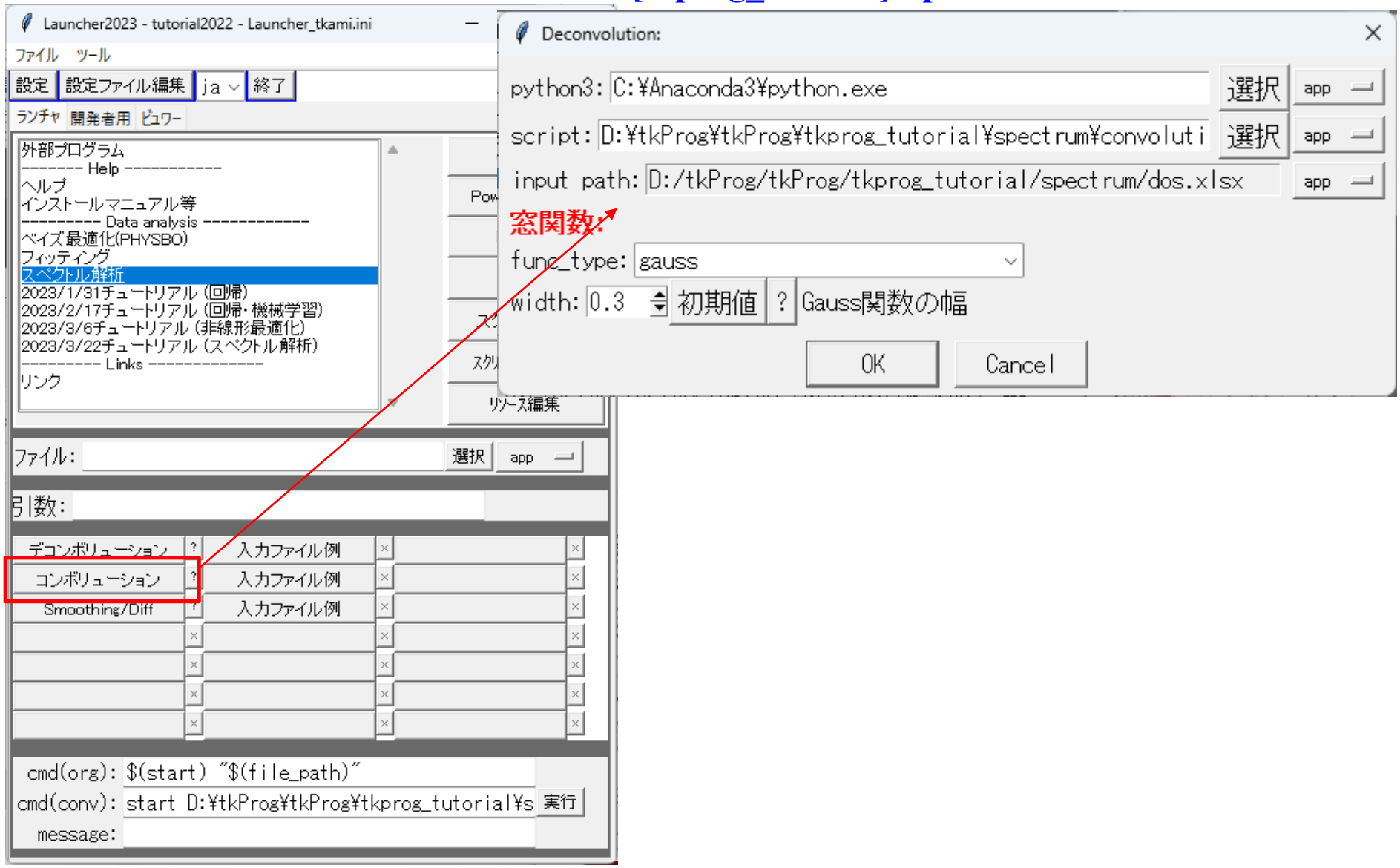
$f(x)$
Intrinsic signal

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)

Program: convolution.py

[tkprog_tutorial]¥spectrum¥dos.xlsx



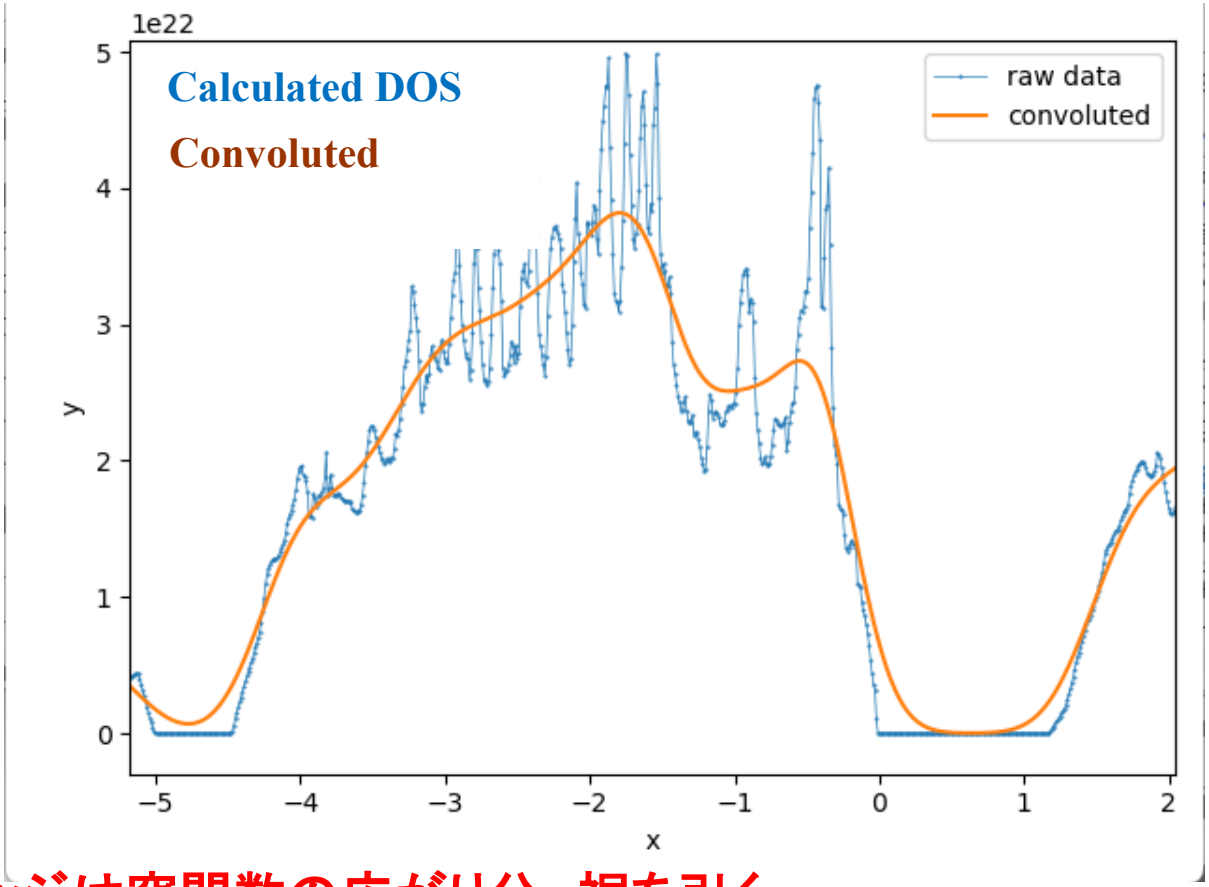
畳み込みによる平滑化 (ぼかし: smearing)

密度汎関数計算で得たa-$InGaZnO_4$の状態密度

問題: スパイキーな構造があり、読み取りにくい

それぞれのデータに幅 0.3 eV の Gauss関数の広がりを加える: 畳み込み

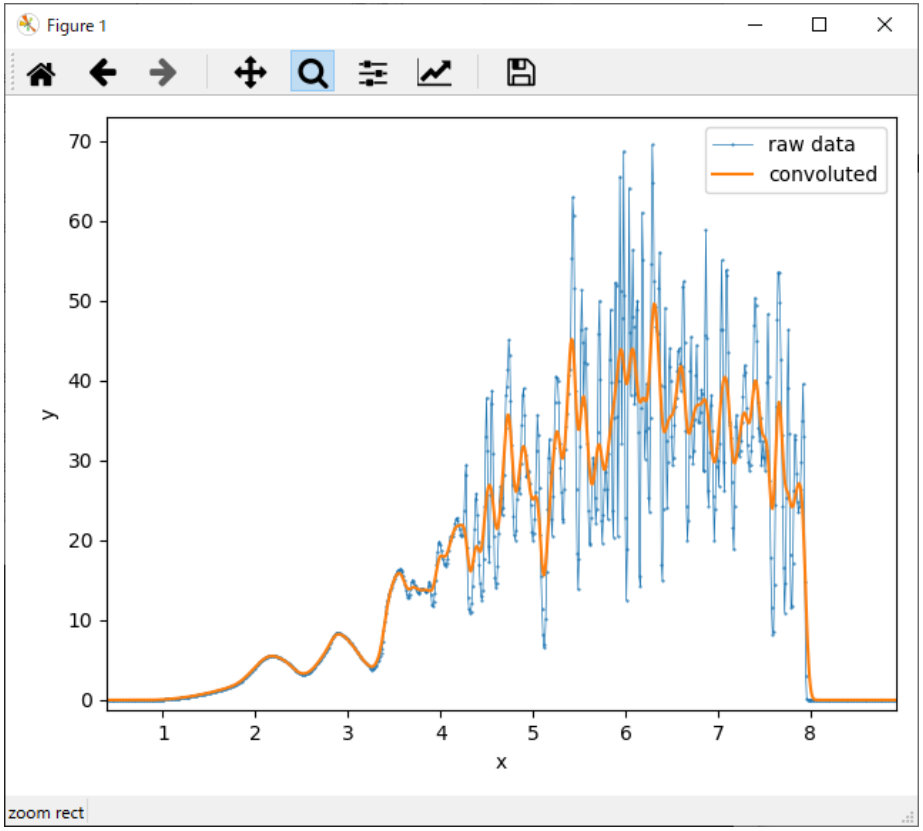
$G(E) = \exp(-[(E - E_0)/w]^2)$ $(w = 0.2\ \mathrm{eV})$



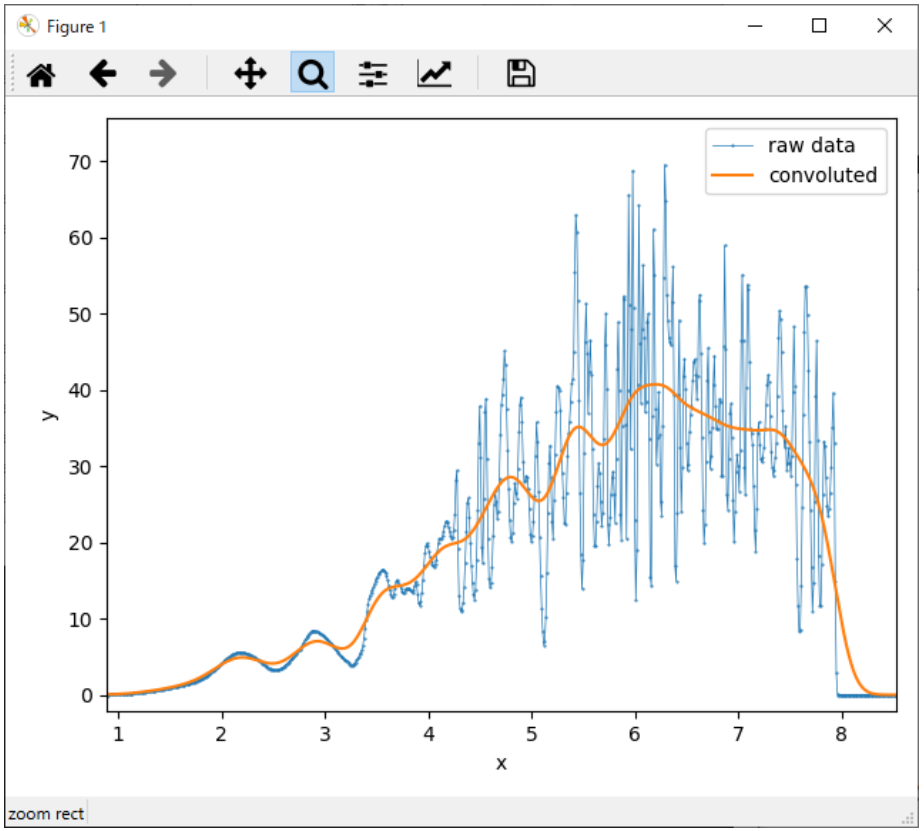
注意: バンドエッジは窓関数の広がり分、裾を引く。
正確なバンドエッジは求められない

畳み込みによる平滑化

w=0.05



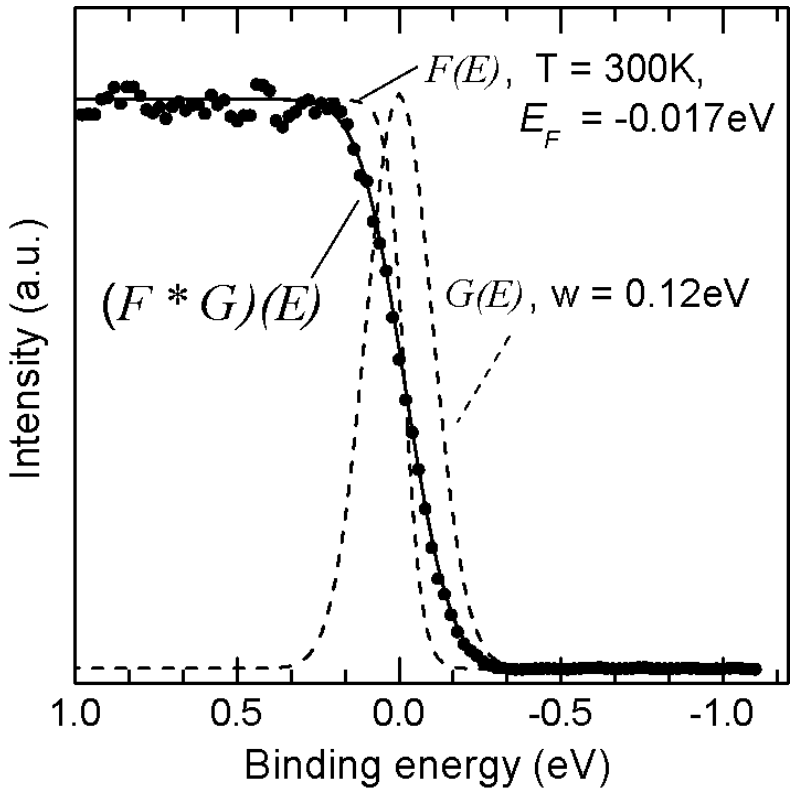
w=0.2



コンボリューション: 装置関数と実測スペクトル

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

例: Au のフェルミ端の紫外光電子スペクトル



試料固有のスペクトル

$S(E)$

装置関数 (窓関数)

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

Fermi-Dirac分布

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

観測されるスペクトル

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

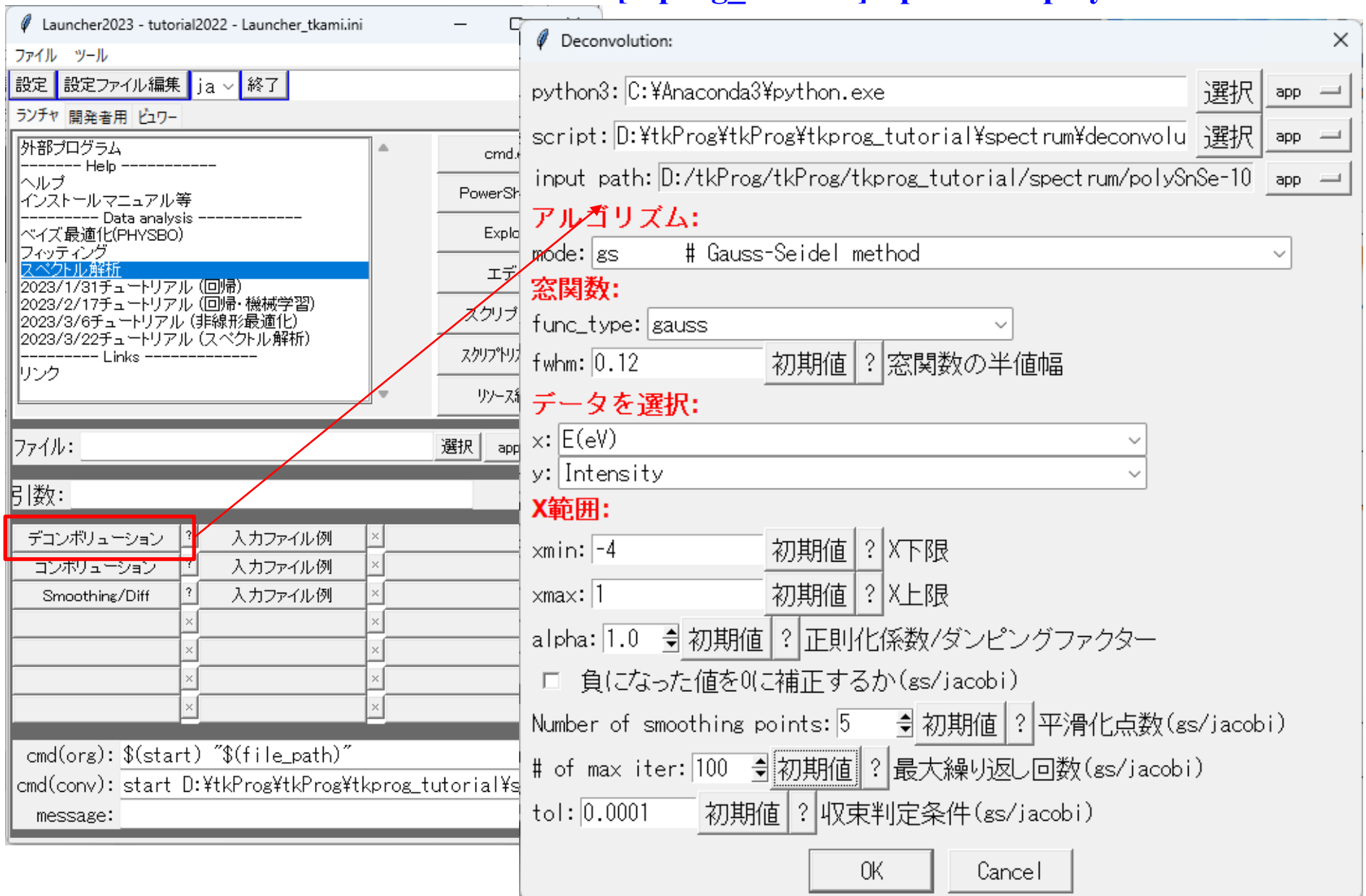
AuではE$_F$付近で $S(E)$ が一定と近似する
$G(E)$ は eq. (1) を $I(x)$ にフィッティングして求められる

$w = 0.12$ eV

***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**

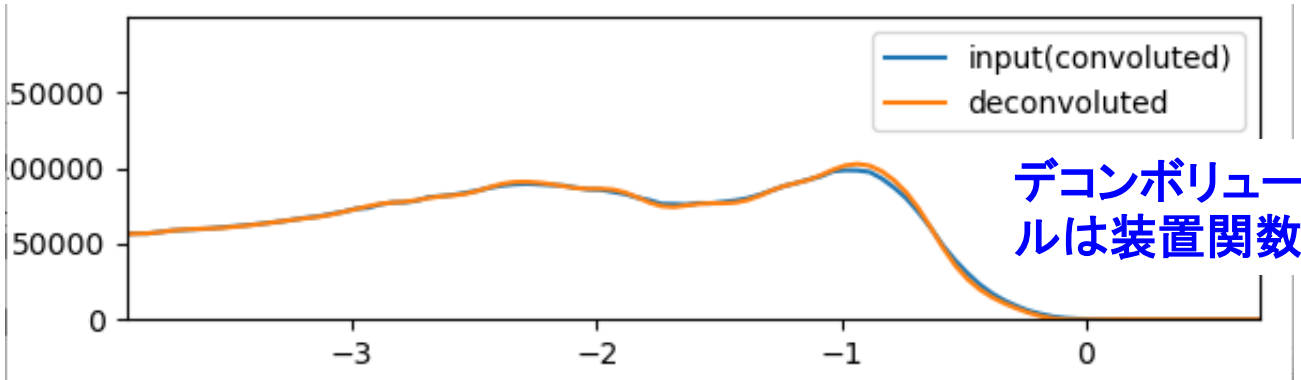
Program: deconvolution.py

[tkprog_tutorial]¥spectrum¥polySnSe-100C.xlsx

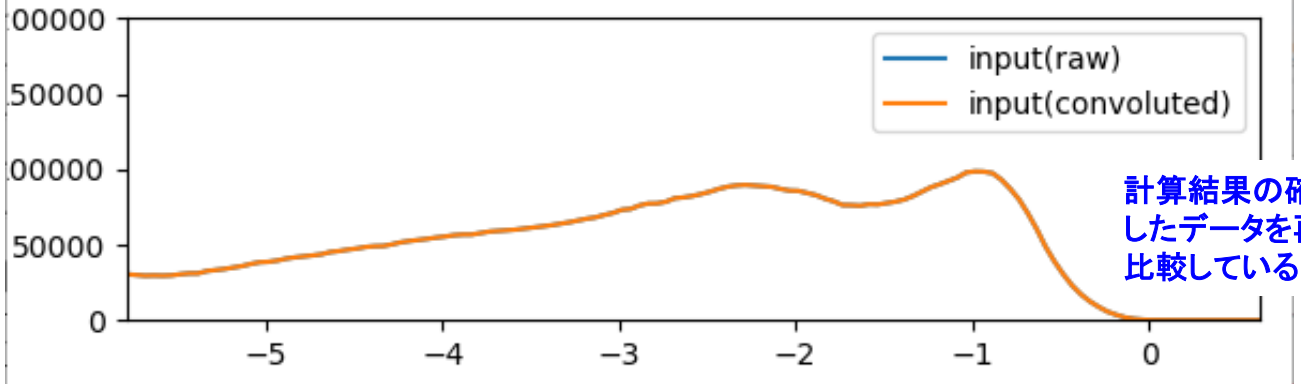


デコンボリューション結果

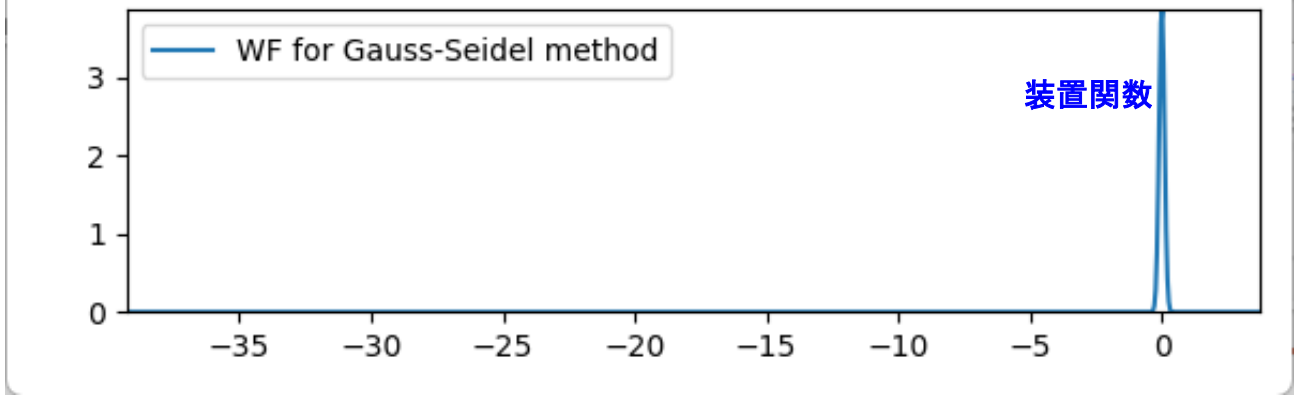
Gauss-Seidel法を使い、繰り返し計算ごとに5点3次多項式平滑化を行う



デコンボリューションしたスペクトルは装置関数分だけ鋭くなる

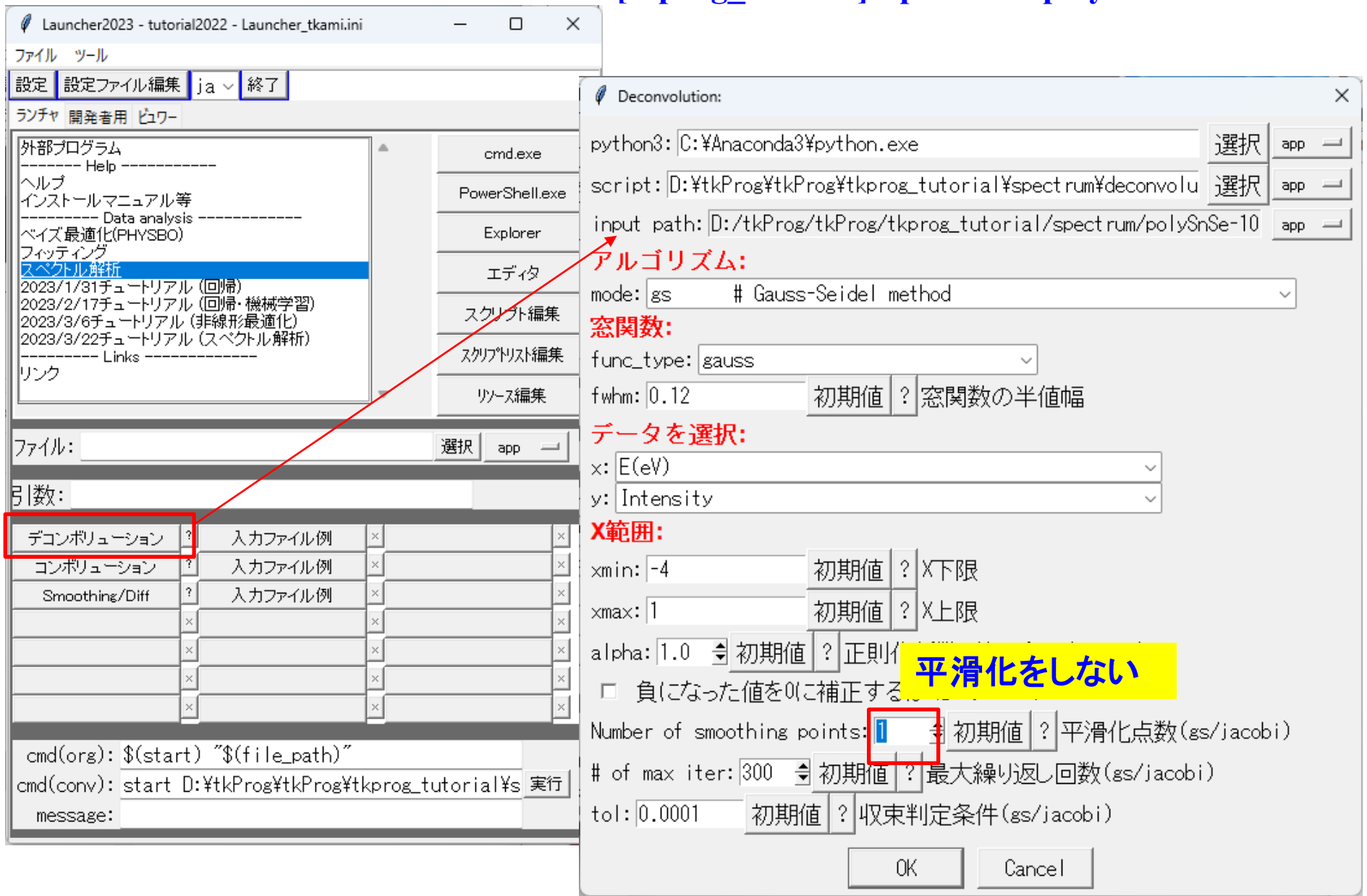


計算結果の確認のため、デコンボリューションしたデータを再コンボリューションして比較している



デコンボリューションの注意

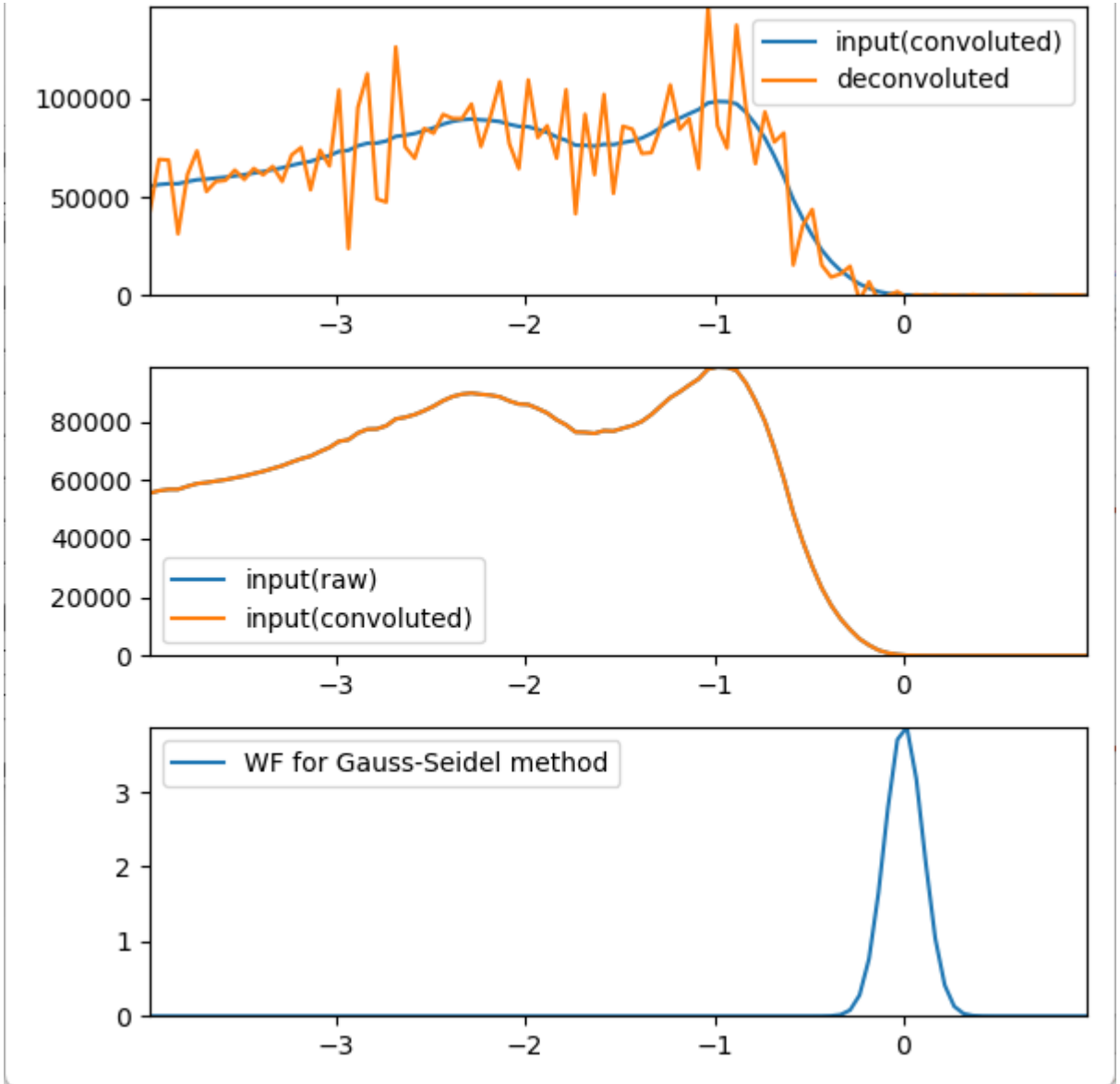
[tkprog_tutorial]¥spectrum¥polySnSe-100C.xlsx



デコンボリューションの注意

平滑化などの操作をしないと、正負に振動するデコンボリューション結果が得られる

ノイズに対して過剰適合した結果



デコンボリューションの解法: FFT

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

フーリエ変換(FT) $$F^*(k) = \int_{-\infty}^{\infty} f^*(x)\exp(ikx)dx$$

逆フーリエ変換 (IFT) $$f(x) = \int_{-\infty}^{\infty} F(k)\exp(-ikx)dk$$

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

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

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

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

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

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

原理的には、*f(x)* はフーリエ変換で $F^*(k)$ をもめ、$F(k) = F^*(k) / G(k)$ を逆フーリエ変換して求められる

=> ノイズなどがあると不安定で解が発散しやすい

デコンボリューションの解法: 行列方程式

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)

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

デコンボリューション $f(x_j)$ は以下の行列式を解けば求められる

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

しかし、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

以下の方法で改善できる:

1. **デコンボリューションの前に、平滑化等でノイズを除去する**
2. **繰り返し法 (Jacobi法や Gauss-Seidel法) で行列方程式を解き、その過程でノイズ削減などの処理を行う**

Jacobi法 / Gauss-Seidel法

大規模行列による線形連立方程式の解:

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

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.

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

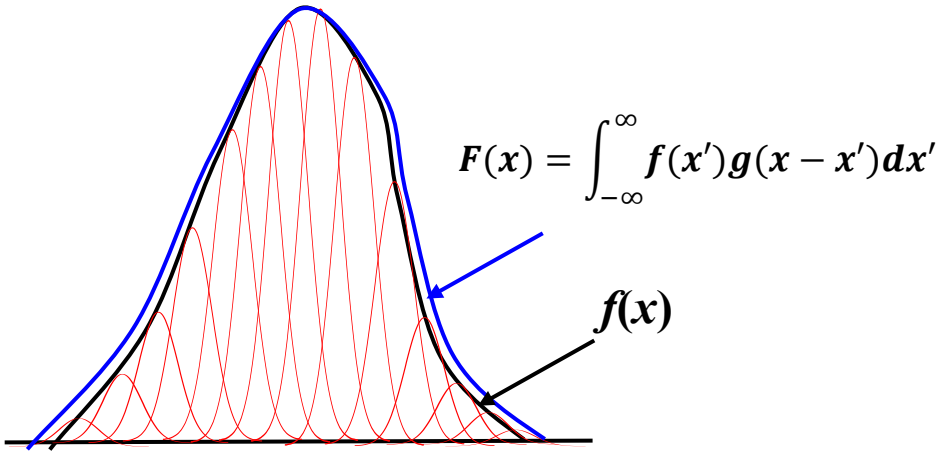
Convergence is better for the Gauss-Seidel method,
While parallelization is more easy for the Jacobi method.

思いつき: Ridge回帰による解法

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

データ点 (x_i, y_i)

$$F(x_i) = \Delta x \sum_j f(x_j)g(x_j - x_i)$$



$g(x_j - x_i)$: 正規化された装置関数

$$\Delta x \sum_j g(x_j - x_i) = 1$$

$f(x_j)$をこのまま求めると大きく振動する (過学習)

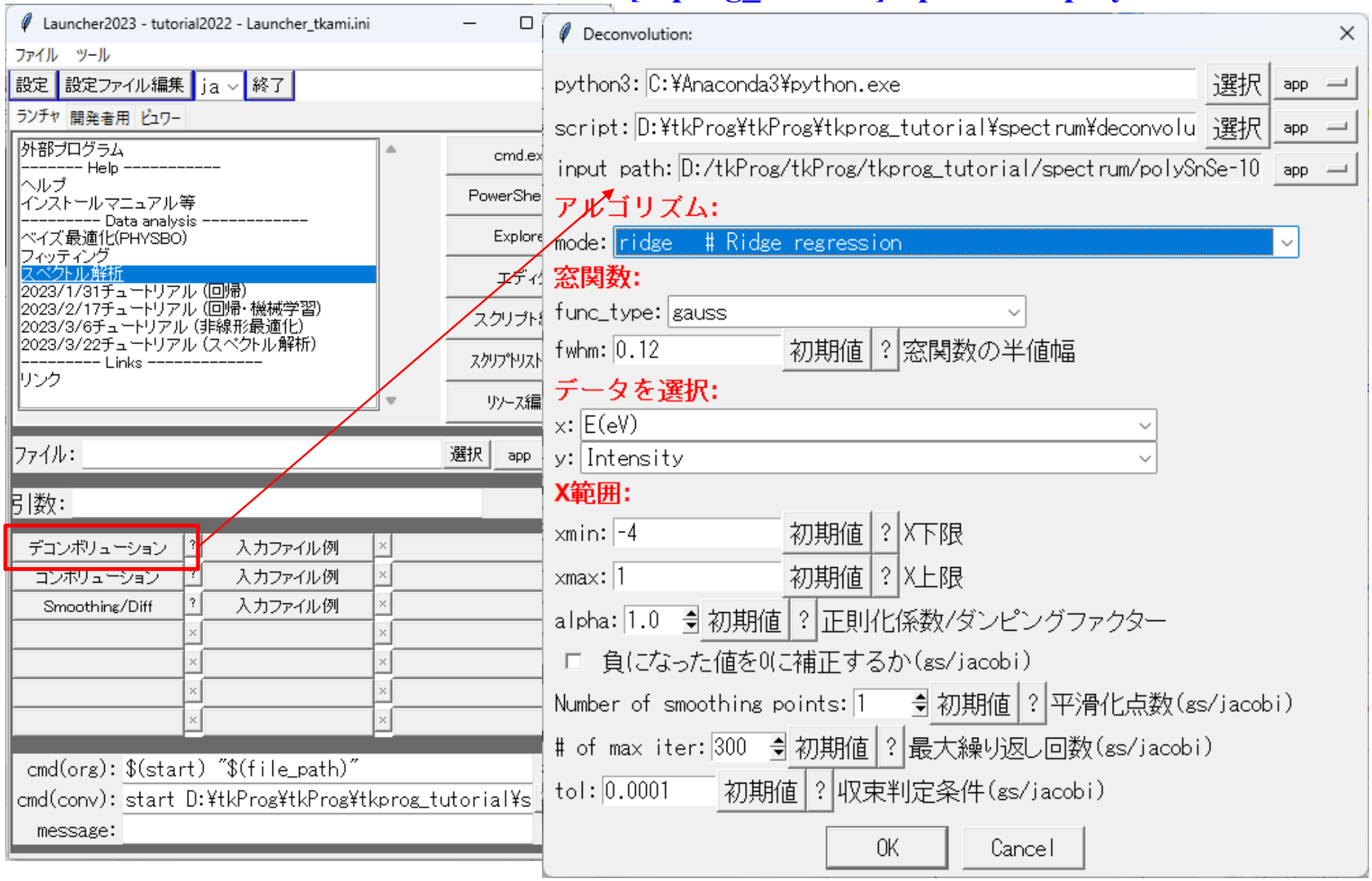
=> Ridge 回帰により振動を抑える

$$\sum_i (y_i - F(x_i))^2 + \alpha \sum_i f(x_j)^2$$

を最小化 (α: L2正則化係数)

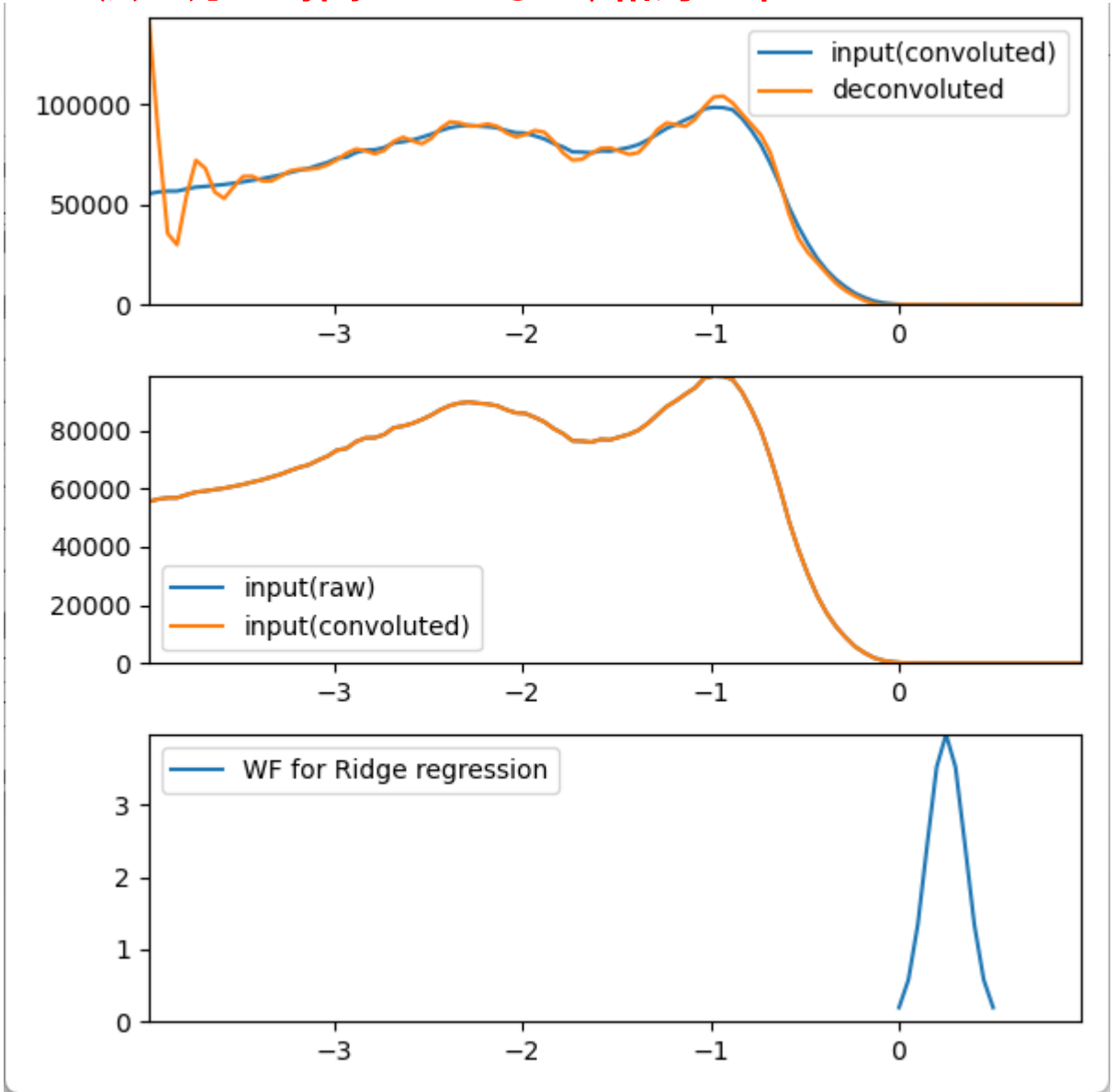
デコンボリューションの注意

[tkprog_tutorial]¥spectrum¥polySnSe-100C.xlsx



Ridge回帰によるデコンボリューション結果

Ridge回帰だと、振動を抑える条件を見つけるのが難しい
Jacobi法やGauss-Seidel法の方が時間がかかるが、結局は早い



Fourier transformation

フーリエ変換

Fourier級数展開

周期: T

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos \frac{2\pi n}{T} t + b_n \sin \frac{2\pi n}{T} t \right)$$

$$a_n = \frac{2}{T} \int_0^T x(t) \cos \frac{2\pi n}{T} t dt$$

$$b_n = \frac{2}{T} \int_0^T x(t) \sin \frac{2\pi n}{T} t dt$$

$$x(t) = \sum_{n=-\infty}^{\infty} c_n \exp \left(i \frac{2\pi n}{T} t \right)$$

$$c_n = \frac{1}{T} \int_0^T x(t) \exp \left(-i \frac{2\pi n}{T} t \right) dt$$

Riemann–Lebesgue lemma (リーマン・ルベーグの補題): $\lim_{n \to \infty} c_n = 0$

フーリエ変換

異なる定義があるので注意

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

フーリエ変換の特徴

- 時間依存データを周波数依存データに変換
- 位置依存データを波数依存データに変換
- 元データの原点は逆空間全体に広がる
- 元データの全体は逆空間の原点に収束する
- フーリエ変換したデータ (FT) を逆変換 (IFT) すると元のデータに戻る

一般関数の線形最小二乗法

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

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

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

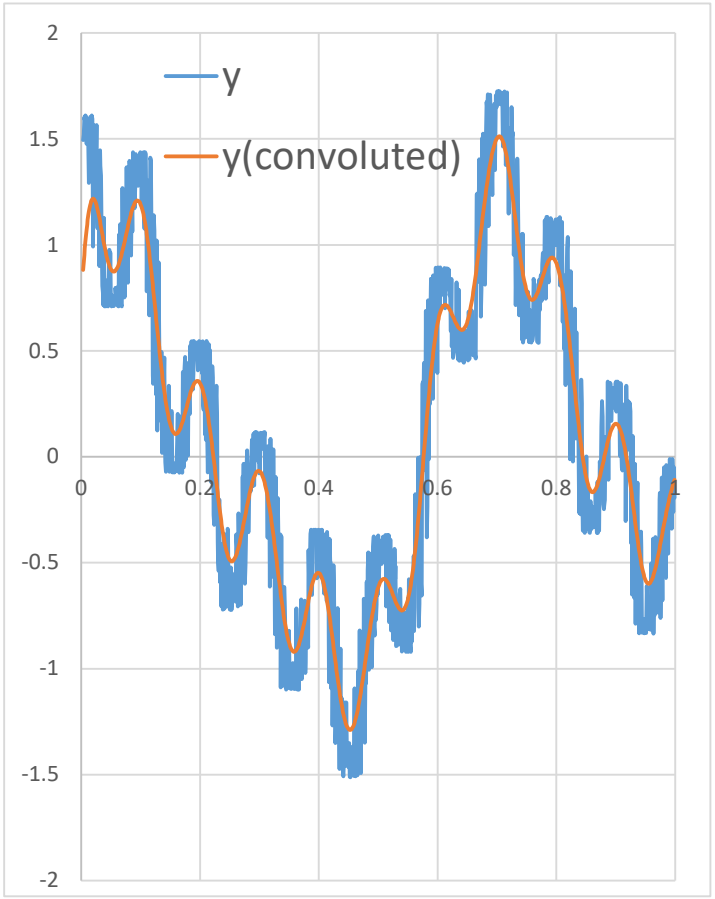
三角関数級数展開への応用例

$f_i(x)=\cos 2\pi f_i x$ **(*i* = odd numbers (奇数))**

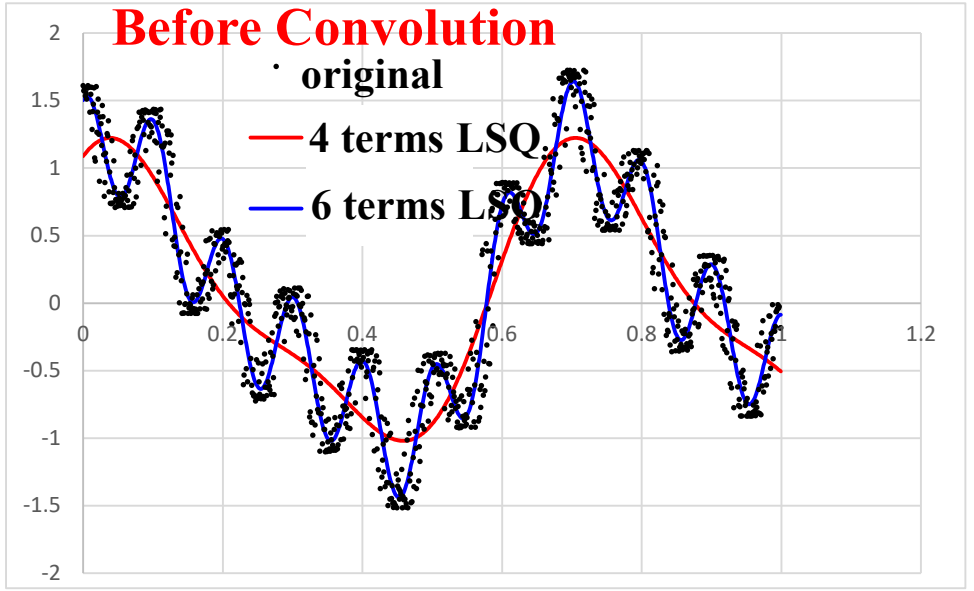
$f_i(x)=\sin 2\pi f_i x$ **(*i* = even numbers (偶数))**

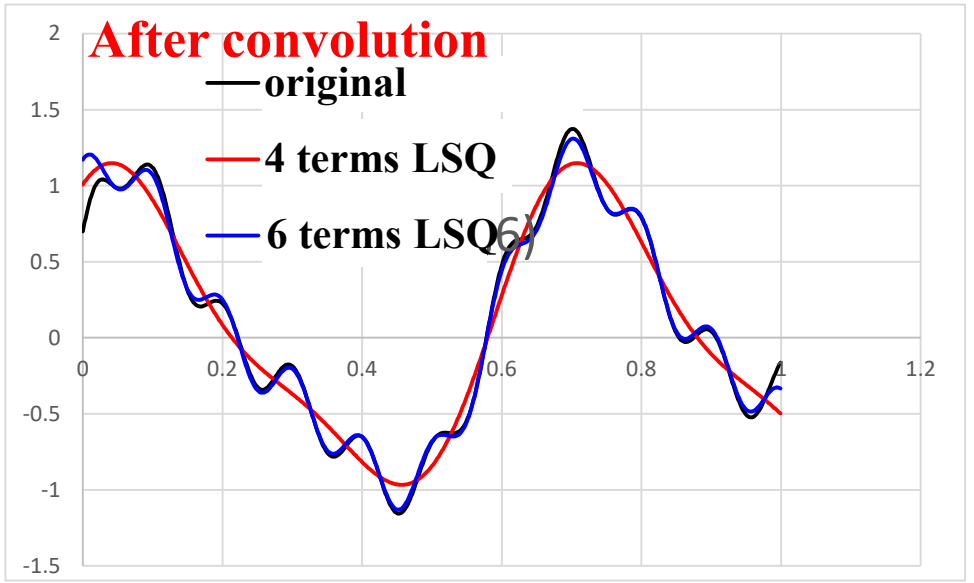
Fourier級数展開への線形最小二乗

f1, p1, A1 = 1.5, pi/4.0, 1.0
f2, p2, A2 = 3.0, pi/3.0, 0.3
f3, p3, A3 = 10.0, pi/6.0, 0.5
x += random(0.03) # noise is simulated by random()
y = A1 * sin(2.0*pi * f1 * x + p1)
+ A2 * sin(2.0*pi * f2 * x + p2)
+ A3 * sin(2.0*pi * f3 * x + p3)
Convolution: Gauss function with w = 0.03



LSQ results





Discrete FT (DFT, 離散フーリエ変換)

Assume $x(t)$ is periodic in the range $[0, T^w]$ and $x(0) = x(T^w)$

$$X(f_k) = T_s^w \sum_{j=0}^{N-1} x(t_j) \exp(-i2\pi f_k \cdot jT^w/N) \qquad T_s^w = T^w/N$$

Usually the coefficient T_s^w is not included for DFT formulations

$$y(f_k) = \sum_{j=0}^{N-1} x(t_j) \exp(-i2\pi kj/N) \qquad f_k = k/T^w$$

DFT can be carried out without many trigonometric function (三角関数) calculations

$$y_k = \sum_{j=0}^{N-1} x_j {w_N}^{kj}$$

$w_N = \exp(-i2\pi/N)$: **Rotation factor (回転因子)**

$$\begin{aligned} {w_N}^{k+1} &= (\cos(-2\pi k/N) + i\sin(-2\pi k/N))(\cos(-2\pi/N) + i\sin(-2\pi/N)) \\ &= \left(\cos(-2\pi k/N)\, w_{N,r} - \sin(-2\pi k/N)\, w_{N,i}\right) \\ &\quad + i\left(\cos(-2\pi k/N)\, w_{N,i} + \sin(-2\pi k/N)\, w_{N,r}\right) \\ &= \left(w_{N,r}^k w_{N,r} - w_{N,i}^k w_{N,i}\right) + i\left(w_{N,r}^k w_{N,i} + w_{N,i}^k w_{N,r}\right) \end{aligned}$$

DFT: Matrix expression (行列表現)

$$y(f_k) = \sum_{j=0}^{N-1} x(t_j) \exp(-i2\pi \cdot k \cdot j/N)$$

$$y_k = \sum_{j=0}^{N-1} x_j w_N^{kj} \qquad w_N = \exp(-i2\pi/N)$$

DFT

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & w_N^{1} & w_N^{2} & w_N^{N-1} \\ \vdots & w_N^{2} & \ddots & \vdots \\ 1 & w_N^{N-1} & \cdots & w_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

Inverse DFT

$$\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & w_N^{-1} & w_N^{-2} & w_N^{-(N-1)} \\ \vdots & w_N^{-2} & \ddots & \vdots \\ 1 & w_N^{-(N-1)} & \cdots & w_N^{-(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix}$$

Using $w_N^{k} = w_N^{k \bmod N}$ and $w_N^{k+N/2} = -w_N^{k}$, only $k = 1 - N/2$ terms should be calculated

Fast FT (FFT, 高速フーリエ変換)

金谷健一, これならわかる応用数学教室, 共立出版社 (2003)

1. The data number must be $N = 2^m$ (m: integer)

2. The identical calculation to DFT, but the calculation cost is proportional only to $N \log N$ (proportional to N^2 for DFT)

3. Simple circuits can implement FFT, easy for parallelization (GPU)

The DFT formulation is written as polynomial by converting $w_N^k = z$

$$y_k = \sum_{j=0}^{N-1} x_j {w_N}^{kj} = \sum_{j=0}^{N-1} x_j z^j$$

$$y_k = x_0 z^0 + x_1 z^1 + x_2 z^2 + \cdots + x_{N-1} z^{N-1}$$

$$= x_0 z^0 + x_2 z^2 + \cdots + x_{N-2} z^{N-2}$$

$$+ z(x_1 z^0 + x_2 z^2 + \cdots + x_{N-1} z^{N-2})$$

Note that the last line equation becomes a polynomial with respect to $z_2 = z^2$ with a half number of the terms

$$y_k = \sum_{j=0}^{N/2-1} x_{2j} {z_2}^j + z \sum_{j=0}^{N/2-1} x_{2j+1} {z_2}^j$$

FFT

金谷健一, これならわかる応用数学教室, 共立出版社 (2003)

$$y_{k,N} = x_0(z^2)^0 + x_2(z^2)^1 + \cdots + x_{N-2}(z^2)^{\frac{N}{2}-1} + z\left(x_1(z^2)^0 + x_2(z^2)^1 + \cdots + x_{N-1}(z^2)^{\frac{N}{2}-1}\right)$$
$$= y_{k,N/2,1} + z y_{k,N/2,2}$$

$$y_{k,N/2,1} = x_0(z^4)^0 + x_4(z^4)^1 + \cdots + x_{N-2}(z^4)^{\frac{N}{4}-1} + (z^2)\left(x_2(z^4)^0 + x_6(z^4)^1 + \cdots + x_{N-3}(z^4)^{\frac{N}{4}-1}\right)$$
$$= y_{k,N/4,1} + (z^2) y_{k,N/4,3}$$

$$y_{k,N/2,2} = x_1(z^4)^0 + x_5(z^4)^1 + \cdots + x_{N-1}(z^4)^{\frac{N}{4}-1} + (z^2)\left(x_3(z^4)^0 + x_7(z^4)^1 + \cdots + x_{N-2}(z^4)^{\frac{N}{4}-1}\right)$$
$$= y_{k,N/4,2} + (z^2) y_{k,N/4,4}$$

$$y_{k,N} = y_{k,N/2,1} + z y_{k,N/2,2}$$
$$y_{k,N/2,1} = y_{k,N/4,1} + z^2 y_{k,N/4,3}$$
$$y_{k,N/2,2} = y_{k,N/4,2} + z^2 y_{k,N/4,4}$$
$$y_{k,N/4,1} = y_{k,N/8,1} + z^4 y_{k,N/8,5}$$
$$y_{k,N/4,2} = y_{k,N/8,2} + z^4 y_{k,N/8,6}$$
$$y_{k,N/4,3} = y_{k,N/8,5} + z^4 y_{k,N/8,7}$$
$$y_{k,N/4,4} = y_{k,N/8,6} + z^4 y_{k,N/8,8}$$

The above is a recursion formula and can be solved from the last two-terms FT to upper equations in the series of the number of terms $2^2, 2^3, \ldots, 2^N$

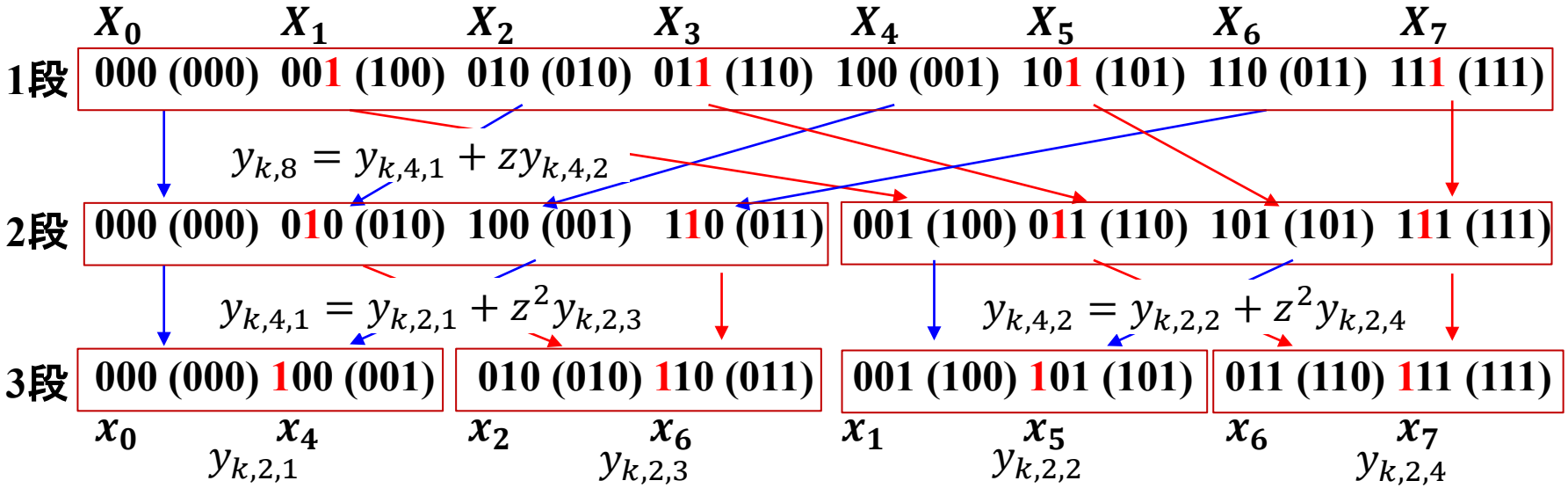
漸化式の形になっているので、最後の項数2のFTから順次 項数$2^2, 2^3, \ldots, 2^N$のFTの計算をすることでFT計算ができる

FFT演算の項順序の変換: ビット反転

N個のデータ列 $x_0 x_1 x_2 \cdots x_{N-1}$ => FT: $X_0 X_1 X_2 \cdots X_{N-1}$
順序数を二進数であらわす

FFTのそれぞれの段階で「奇数番目のデータを後半にずらす」操作をする
=> 順序数の右から「段階数に対応するビットが 1 のデータを後半にずらす」
=> 順序数の変換がビット反転に対応する

最初のデータの並び順 (カッコ内は順序数のビット反転)



バタフライ演算

FFTの和を取る順番は x_i の並び順と変わる。
最初の順序数の二進数表現 (カッコ内の数字) をビット反転 (カッコ外の数字) してソートすると、その順序でFFTの和をとれる

Logical operations (bitwise operations)
(論理演算, ビット演算)

Logical NOT (Bitwise inversion) (論理否定, ビット反転)

NOT 0 = 1; NOT 1 = 0

python: ~x, not x *~1 == 0, ~0 == 1*

Logical AND (論理積)

0 AND 0 = 0; 1 AND 0 = 0
0 AND 1 = 0; 1 AND 1 = 1

python: x & y, x and y *1 & 1 == 1*

Logical OR (論理和)

0 OR 0 = 0; 1 OR 0 = 1
0 OR 1 = 1; 1 OR 1 = 1

python: x | y, x or y

Logical Exclusive OR (排他的論理和)

0 XOR 0 = 0; 1 XOR 0 = 1
0 XOR 1 = 1; 1 XOR 1 = 0

python: x ^ y, x xor y

Bit shift (*n* bit shift)

python: a << n, a >> n *0b0001 << 2 == 0b0100*
0b0110 >> 1 == 0b0011

Bit reversal (ビット列反転)

Note: **bit reversal (ビット列反転) != bitwise inversion (ビット反転) (~x, not x)**

bit_reverse.py

```
def bit_reverse(val):
    ret = 0
    while 1:
        v0 = val & 0b001
        ret = ret | v0
        val = val >> 1

        if val == 0:
            break
        else:
            ret = ret << 1

    return ret
```

val = 11001_2 を例に

ret = 0 # ビット反転値を0で初期化

1. **v0** = val & 1_2 => 11001_2 & 001_2 **= 1** # 第1桁のビット値を v0 に保存
2. **ret** = ret | v0 => 0 | 1 **= 1_2** # retの第一桁に v0 を設定
3. **val** = val >>1 => 11001_2 >> 1 **= 1100_2**
   # 一桁右にビットシフトし、valの2桁目を第1桁に移動
4. val が 0 の場合、処理するbitが残っていないので
   ループを終了
5. val が 0でない場合、ret を1ビットシフトし、2.で ret の第1位に設定した v0 を左にずらす。
   ret = ret << 1 => 1_2 << 1 **= 10_2**
   1.に戻って繰り返し
6. **v0** = val & 1_2 => 1100_2 & 001_2 = 0 # 第1桁のビット値を v0 に保存
7. **ret** = ret | v0 => 10_2 | 1 **= 10_2** # retの第一桁に v0 を設定
8. **val** = val >>1 => 1100_2 >> 1 **= 110_2**
9. **ret** = ret << 1 => 10_2 << 1 **= 100_2**
   1.に戻って繰り返し
10. **v0** = val & 1_2 => 110_2 & 001_2 **= 0**
11. **ret** = ret | v0 => 100_2 | 1 = 100_2
12. **val** = val >>1 => 110_2 >> 1 **= 11_2**
13. **ret** = ret << 1 => 100_2 << 1 **= 1000_2**
    1.に戻って繰り返し
14. **v0** = val & 1_2 => 11_2 & 1_2 **= 1**
15. **ret** = ret | v0 => 1000_2 | 1 = 1001_2
16. **val** = val >>1 => 11_2 >> 1 **= 1_2**
17. **ret** = ret << 1 => 1001_2 << 1 **= 10010_2**
    1.に戻って繰り返し
18. **v0** = val & 1_2 => 1_2 & 1_2 **= 1**
19. **ret** = ret | v0 => 10010_2 | 1 = 10011_2
20. **val** = val >>1 => 1_2 >> 1 **= 0_2** => ループ終了 解: **ret = 10011_2**

Bitwise operation can be replaced with other op

Usually bitwise operations are faster, but it is not the case for python …

python bit_reverse_compare.py 10011000111101011011111011 1000000

measure time to reverse $10011000111101011011111011_2$ for 1000000 times

by bitwise operation : 6.265091180801392 s

without bitwise operation: 4.462110280990601 s

bit_reverse_compare.py

```
def bit_reverse(val):
    ret = 0
    while 1:
        v0 = val & 0b001
        ret = ret | v0
        val = val >> 1

        if val == 0:
            break
        else:
            ret = ret << 1

    return ret
```

bit_reverse_compare.py

```
def bit_reverse_nobitop(val):
    ret = 0
    while 1:
        v0 = val % 2        # save the final bit to v0
        ret = ret + v0      # put v0 to the final bit of ret
        val = val // 2      # bit shift for next iteration

        if val == 0:
            break
        else:
            ret = ret * 2   # bit shift ret to left

    return ret
```

Comparison: Calculation time by python

Usage: python dft.py ndata
ex: python dft.py 1024
python dft.py 2048

DFT1: DFT using rotation factor
DFT2: DFT not using rotation factor (calculate sin/cos every time)
FFT : numpy.fft.fft()

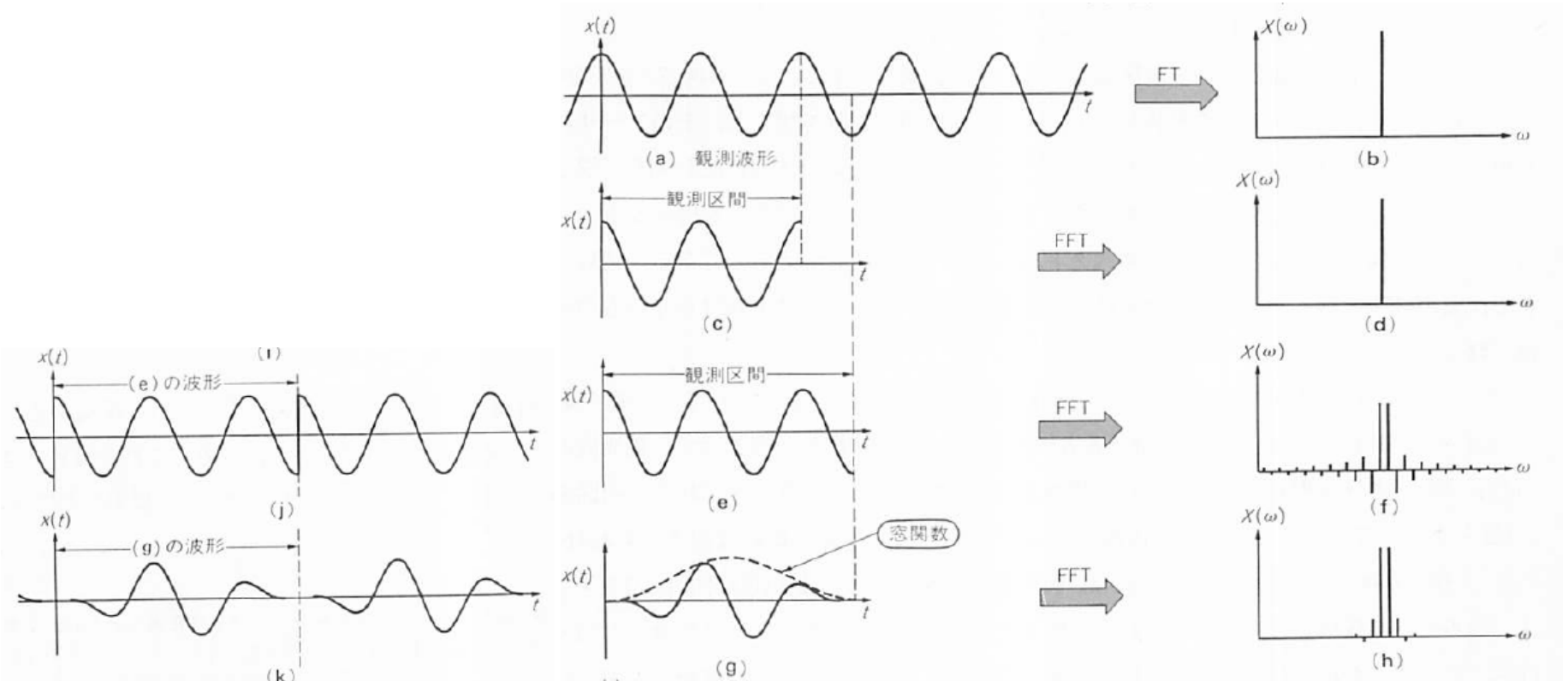
Time for DFT/FFT (sec)

N	DFT1	DFT2	FFT	N log N
1024	1.32	2.41	1.87e-5	3080
2048	5.59	10.3	3.54e-5	6780
4096	23.6	47.7	6.62e-5	14800
8192	97.3	165	16.1e-5	32100

Problems of DFT/FFT

南茂夫, 科学計測のための波形データ処理, CQ出版社 (1986)

- **Usually FT needs integration from -∞ to ∞, but DFT/FFT reduces data to finite range => Loss of data**
  ***ex.*: Fourier charge analysis by XRD gives ghost peaks and fringes**
- **Original data include noise/errors, giving rise to extra frequency peak**
- **Artificial periodicity required for DFT/FFT gives rise to artefact frequency peaks => can be suppressed by Hanning Window (窓関数), but it may also give extra peaks**



Maximum entropy method (MEM, 最大エントロピー法)

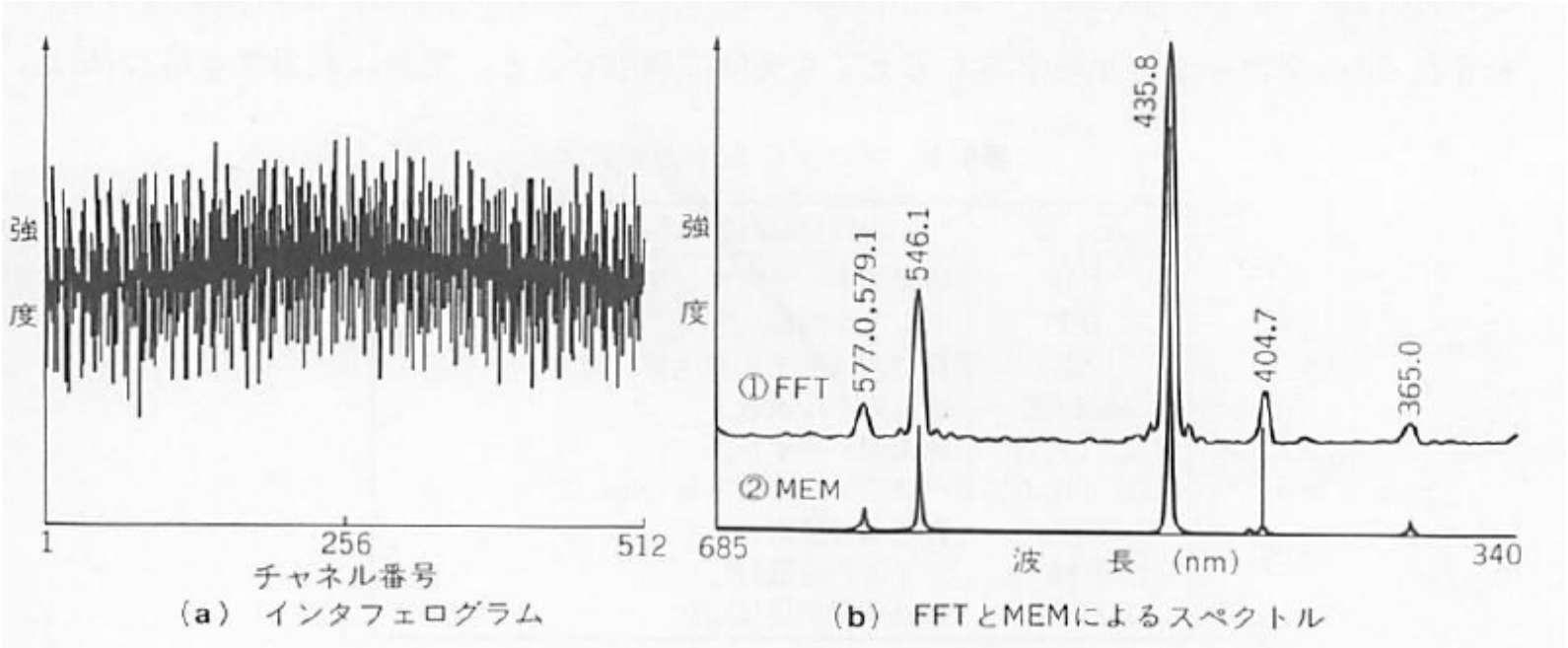
南茂夫, 科学計測のための波形データ処理, CQ出版社 (1986)

Concept of MEM

- Assume the lost data would have some constraints
- Use the concept of 'information entropy' and maximize it to estimate the spectrum
- Akaike's autoregressive model (赤池による自己回帰モデル) => identical to MEM
  The order of the autoregressive model *m* must be determined
  - So as to minimize Final Prediction Error (最終予測誤差)
- Algorisms: Burg method, etc

Features

- Sharper spectrum than FFT
- Less ghost peaks and fringes



MEM-Rietveld analysis

坂田誠 日本結晶学会誌 **30**, 135 (1988)

Charge density calculated from structure factors $\tau'_i = \tau_i / \sum \tau_i$

Charge density calculated from structure model $\rho'_i = \rho_i / \sum \rho_i$

Constrained entropy: $S = -\sum \rho'_i \ln \frac{\rho'_i}{\tau'_i}$

=> smoothing ρ' and suppress fringes and ghost peaks

Minimize the structure factor residual $C = \sum \frac{\left|F_{\mathrm{cal}}{}^{hkl} - F_{\mathrm{obs}}{}^{hkl}\right|^2}{\sigma_{hkl}{}^2}$

Maximize constrained entropy $Q(\lambda) = -\sum \rho'_i \ln \frac{\rho'_i}{\tau'_i} - \frac{\lambda}{2} \sum \frac{\left|F_{\mathrm{cal}}{}^{hkl} - F_{\mathrm{obs}}{}^{hkl}\right|^2}{\sigma_{hkl}{}^2}$

=> ρ = exp(ln τ + difference Fourier (差フーリエ) term)

When converged to $F_{\mathrm{cal}} = F_{\mathrm{obs}}$, ρ = τ will be achieved

Kramers-Kronig Transformation

クラマースークローニッヒ (KK) の関係式

因果律が成立していれば
(現在の状態が、過去の履歴の蓄積で決定していれば)、
線形応答の範囲で、周波数応答関数 $\varepsilon(\omega)$ の実部と虚部には
以下のKramers-Kronighの関係が成立する

$$\varepsilon_r = 1 + \frac{1}{\pi} P\int_{-\infty}^{\infty} \frac{\omega' \varepsilon_i(\omega')}{\omega'^2 - \omega^2} d\omega' \left(= 1 + \frac{2}{\pi} P\int_{0}^{\infty} \frac{\omega' \varepsilon_i(\omega')}{\omega'^2 - \omega^2} d\omega' \right)$$

$$\varepsilon_i = -\frac{1}{\pi} P\int_{-\infty}^{\infty} \frac{\varepsilon_r(\omega') - 1}{\omega'^2 - \omega^2} d\omega' \left(= -\frac{2}{\pi} P\int_{0}^{\infty} \frac{\varepsilon_r(\omega') - 1}{\omega'^2 - \omega^2} d\omega' \right)$$

インパルス応答が実数の
場合、カッコ内の式が使える

P: 積分の主値 $P\int_0^{\infty} d\omega' \equiv \lim_{\delta \to 0} \left(\int_0^{\omega-\delta} d\omega' + \int_0^{\omega+\delta} d\omega' \right)$

注意 : 主値積分は極限の取り方によって値が変わる
=> 必ず ωの両側から同じように極限を取る

クラマースークローニッヒの関係式

Kramers-Kronig relation

$$\varepsilon_r = 1 + \frac{2}{\pi} P \int_0^\infty \frac{\omega' \varepsilon_i(\omega')}{\omega'^2 - \omega^2} d\omega'$$

$$\varepsilon_i = -\frac{2}{\pi} P \int_0^\infty \frac{\varepsilon_r(\omega') - 1}{\omega'^2 - \omega^2} d\omega'$$

P: Principal value of the integral

$$P\int_0^\infty d\omega' \equiv \lim_{\delta \to 0}\left(\int_0^{\omega-\delta} d\omega' + \int_0^{\omega+\delta} d\omega' \right)$$

The above equation is derived from Cauchy integral $\alpha(\omega) = \frac{1}{\pi i} P \int_0^\infty \frac{\alpha(s)}{s-\omega} ds$ that is valid for complex functions $\alpha(\omega)$ satisfying $\lim_{|\omega| \to \infty} \alpha(\omega) = 0$

KK relation: Refrectivity spectrum and phase
反射率と位相

$$r^*(\nu)=\sqrt{R(\nu)}e^{i\theta(\nu)} \qquad \ln r^*(\nu)=\ln R^{1/2}+i\theta(\nu)$$

にKK変換を当てはめる

$$\theta(\nu)=-\frac{1}{2\pi}P\int_0^\infty \frac{\ln R(\nu')}{\nu'^2-\nu^2}d\nu'=-\frac{1}{2\pi}\int_0^\infty \ln\frac{|\nu'+\nu|}{|\nu'-\nu|}\frac{dR(\nu')}{d\nu'}d\nu'$$

Optical spectrum (誘電関数ε*, 吸収係数α)

$$\mathcal{H} = \mathcal{H}_0 - \boldsymbol{er}\cdot\boldsymbol{E}$$

$$\varepsilon_1(\omega) = 1 + 4\pi\sum_j \frac{e^2\left|T_{0j}\right|^2}{\hbar}\frac{2\omega_j}{\omega_j{}^2 - \omega^2}$$

$$T_{ij} = \left\langle \Psi_i \left| \mathbf{r} \right| \Psi_j \right\rangle = \int \Psi_i{}^* \mathbf{r} \Psi_j d\mathbf{r}$$

Kramers-Kronig transformation

$$\varepsilon_2(\omega) = \frac{4\pi Ne^2}{m}\sum_j f_j \pi\delta\left(\omega^2 - \omega_j{}^2\right)$$

$$= \frac{4\pi Ne^2}{m}\sum_j f_j \frac{\pi}{2\omega}\left[\delta\left(\omega - \omega_j\right) + \delta\left(\omega + \omega_j\right)\right]$$

$$n(\omega) - i\kappa(\omega) = \sqrt{\varepsilon_1(\omega) - i\varepsilon_1(\omega)}$$

$$\alpha(\omega) = \frac{4\pi}{\lambda}\kappa(\omega)$$

KK transformation: Numerical approach

$$\theta(\nu_g) = -\frac{2\nu_g}{\pi}\int_0^{\infty} \frac{\ln\sqrt{R(\nu)/R(\nu_g)}}{\nu^2 - \nu_g^{\,2}}\,d\nu$$

$$\theta_{med}(\nu_g) = \frac{4\nu_g}{\pi}\Delta\nu \sum_{i=odd\ or\ even}^{i<=nData} \frac{\ln\sqrt{R_i}}{\nu_i^{\,2} - \nu_g^{\,2}}$$

Numerical integration for given data

$$\theta_{high}(\nu_g) = -\frac{\ln\sqrt{R_{high}}}{\pi}\ln\frac{\nu_{\max} + \nu_g}{\nu_{\max} - \nu_g}$$

Extrapolation to high *f*

$$\theta_{low}(\nu_g) = -\frac{\ln\sqrt{R_{low}}}{\pi}\ln\frac{\nu_g - \nu_{\min}}{\nu_{\min} + \nu_g}$$

Extrapolation to low *f*

$$n(\nu) = \frac{1 - R(\nu)}{1 + R(\nu) - 2\sqrt{R(\nu)}\cos\theta}$$

$$k(\nu) = \frac{2\sqrt{R(\nu)}\sin\theta}{1 + R(\nu) - 2\sqrt{R(\nu)}\cos\theta}$$

光学スペクトルのKK変換: 外挿問題

Fourier変換と同様に、測定周波数外の情報がないことが問題
測定周波数外のデータは結果に大きく影響する

0.1eVにおける分散を正確に求める場合、
少なくとも4~5eVまでの測定が必要

・高エネルギー領域: ν^{-4}で外挿するのが一般的
・知りたい領域が0.1eV以下:
  ・金属の低エネルギー領域: Drude反射率で外挿
  ・半導体・誘電体の低エネルギー領域:
    静的誘電率ε_0から求めた反射率 で外挿 $\left(\sqrt{\varepsilon_0}-1\right)^2/\left(\sqrt{\varepsilon_0}+1\right)^2$

・知りたい領域が1eV程度まで:
  フォノン分散の始まる0.1eV以下は考慮する必要はない

・半導体・誘電体の低エネルギー領域:
  高周波誘電率ε_∞から求めた反射率 で外挿 $\left(\sqrt{\varepsilon_\infty}-1\right)^2/\left(\sqrt{\varepsilon_\infty}+1\right)^2$