

**Fourier transform**

フーリエ変換

# Fourier series expansion (Fourier級数展開)

Period:  $T$

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

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

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

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

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

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

# Fourier transform

Take limit to  $T \Rightarrow \infty$  for Fourier series expansion

$$\left\{ \begin{array}{l} \text{FT} \quad F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(i\omega t) dt \\ \text{IFT} \quad f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \exp(-i\omega t) d\omega \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{FT} \quad F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(i2\pi ft) dt \\ \text{IFT} \quad f(t) = \int_{-\infty}^{\infty} F(\omega) \exp(-i2\pi ft) d\omega \end{array} \right.$$

## Features of Fourier transform

- 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

**Width  $W$  Gauss func is converted to width  $W^{-1}$  Gauss func**

- **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

# LSQ for general function

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

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

## Application to sin / cos expansion

$$f_i(x) = \cos 2\pi f_i x \quad (i = \text{odd numbers (奇数)})$$

$$f_i(x) = \sin 2\pi f_i x \quad (i = \text{even numbers (偶数)})$$

# LSQ for Fourier series expansion

$f_1, p_1, A_1 = 1.5, \pi/4.0, 1.0$

$f_2, p_2, A_2 = 3.0, \pi/3.0, 0.3$

$f_3, p_3, A_3 = 10.0, \pi/6.0, 0.5$

$x += \text{random}(0.03)$  # noise is simulated by random()

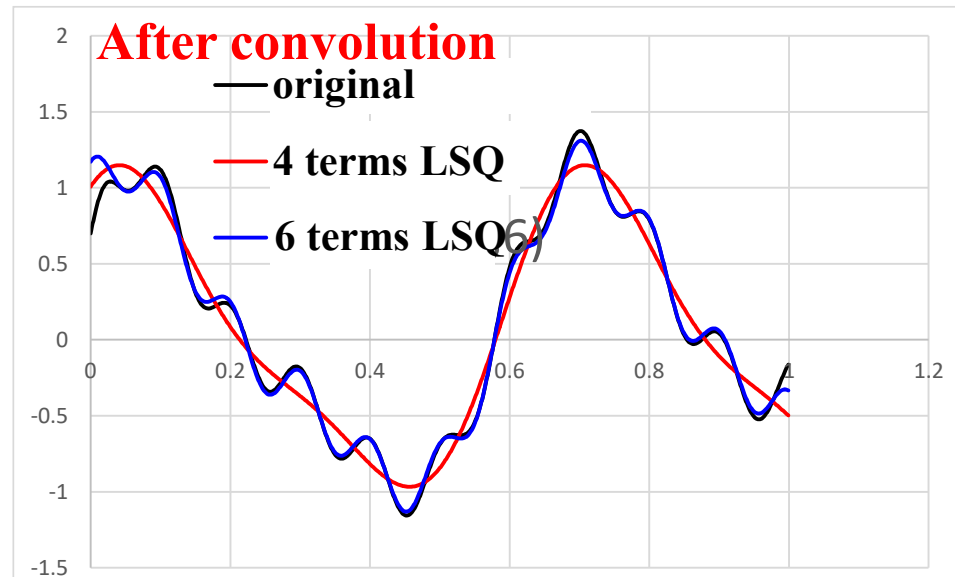
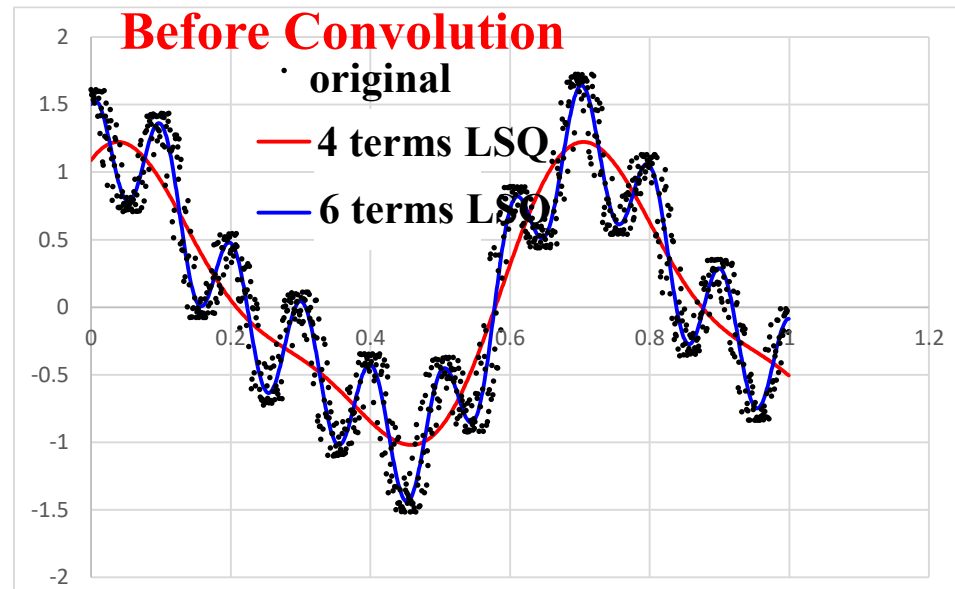
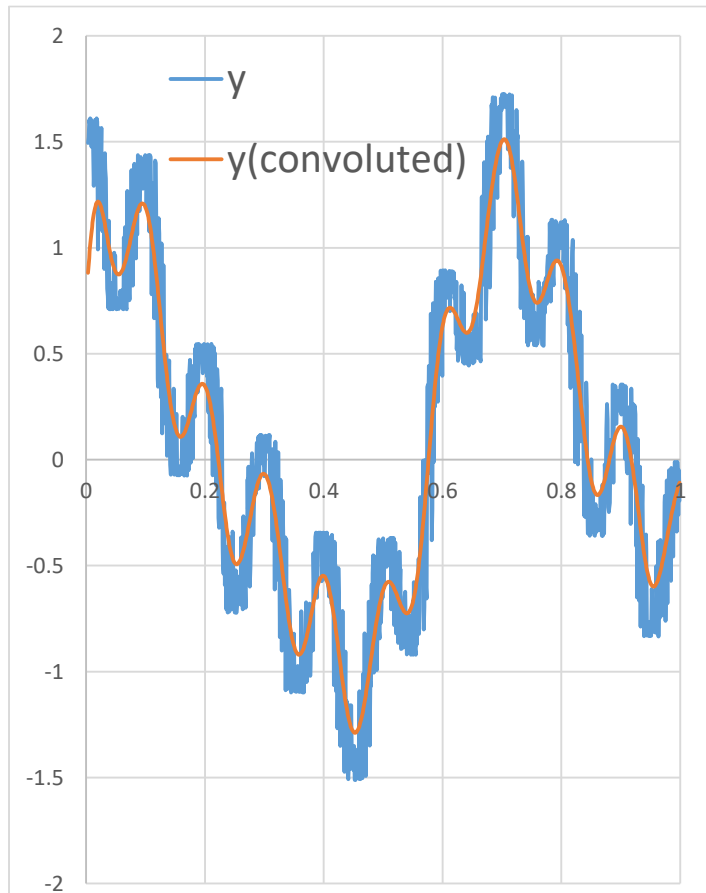
$y = A_1 * \sin(2.0 * \pi * f_1 * x + p_1)$

$+ A_2 * \sin(2.0 * \pi * f_2 * x + p_2)$

$+ A_3 * \sin(2.0 * \pi * f_3 * x + p_3)$

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  $x(0) = x(T^w)$

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

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

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

**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)

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

# DFT: Matrix expression (行列表現)

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

$$\mathbf{y}_k = \sum_{j=0}^{N-1} x_j w_N^{kj}$$

$$w_N = \exp(-i2\pi/N)$$

## 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 Fourier Transform (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)

**FFT: Efficient algorithm to compute the same result as DFT**

**Restriction for radix-2 FFT:**

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

**Merit:**

- 1. Same result as DFT**
- 2. The calculation cost is  $O(N \log N)$  instead of  $O(N^2)$  for direct DFT**
- 3. FFT consists of simple butterfly operations and is suitable for parallel computation (GPU).**

# 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)

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

$$y_k = \sum_{j=0}^{N-1} x_j w_N^{kj} = \sum_{j=0}^{N-1} x_j z^j \quad w_N = \exp(-i2\pi/N): \text{Rotation factor}$$

$$\begin{aligned} y_k &= x_0 z^0 + x_1 z^1 + x_2 z^2 + \cdots + x_{N-1} z^{N-1} \\ &= x_0 z^0 + x_2 z^2 + \cdots + x_{N-2} z^{N-2} \\ &\quad + z(x_1 z^0 + x_3 z^2 + \cdots + x_{N-1} z^{N-2}) \end{aligned}$$

**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)

$$\begin{aligned}y_{k,N} &= x_0(z^2)^0 + x_2(z^2)^1 + \cdots + x_{N-2}(z^2)^{\frac{N}{2}-1} + z \left( x_1(z^2)^0 + x_3(z^2)^1 + \cdots + x_{N-1}(z^2)^{\frac{N}{2}-1} \right) \\ &= y_{k,N/2,1} + z y_{k,N/2,2} \\ y_{k,N/2,1} &= x_0(z^4)^0 + x_4(z^4)^1 + \cdots + x_{N-2}(z^4)^{\frac{N}{4}-1} + (z^2) \left( x_2(z^4)^0 + x_6(z^4)^1 + \cdots + x_{N-3}(z^4)^{\frac{N}{4}-1} \right) \\ &= y_{k,N/4,1} + (z^2) y_{k,N/4,3} \\ y_{k,N/2,2} &= x_1(z^4)^0 + x_5(z^4)^1 + \cdots + x_{N-1}(z^4)^{\frac{N}{4}-1} + (z^2) \left( x_3(z^4)^0 + x_7(z^4)^1 + \cdots + x_{N-2}(z^4)^{\frac{N}{4}-1} \right) \\ &= y_{k,N/4,2} + (z^2) y_{k,N/4,4}\end{aligned}$$

$$y_{k,N} = y_{k,N/2,1} + z y_{k,N/2,2}$$

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

$$y_{k,N/2,2} = y_{k,N/4,2} + z^2 y_{k,N/4,4}$$

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

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

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

$$y_{k,N/4,4} = y_{k,N/8,6} + z^4 y_{k,N/8,8}$$

**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  $2^2, 2^3, \dots, 2^N$**

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 $2^2, 2^3, \dots, 2^N$ のFTの計算をすることでFT計算ができる

# Data swap in the FFT procedure

$N$  data series  $x_0x_1x_2 \cdots x_{N-1} \Rightarrow$  FT:  $X_0X_1X_2 \cdots X_{N-1}$

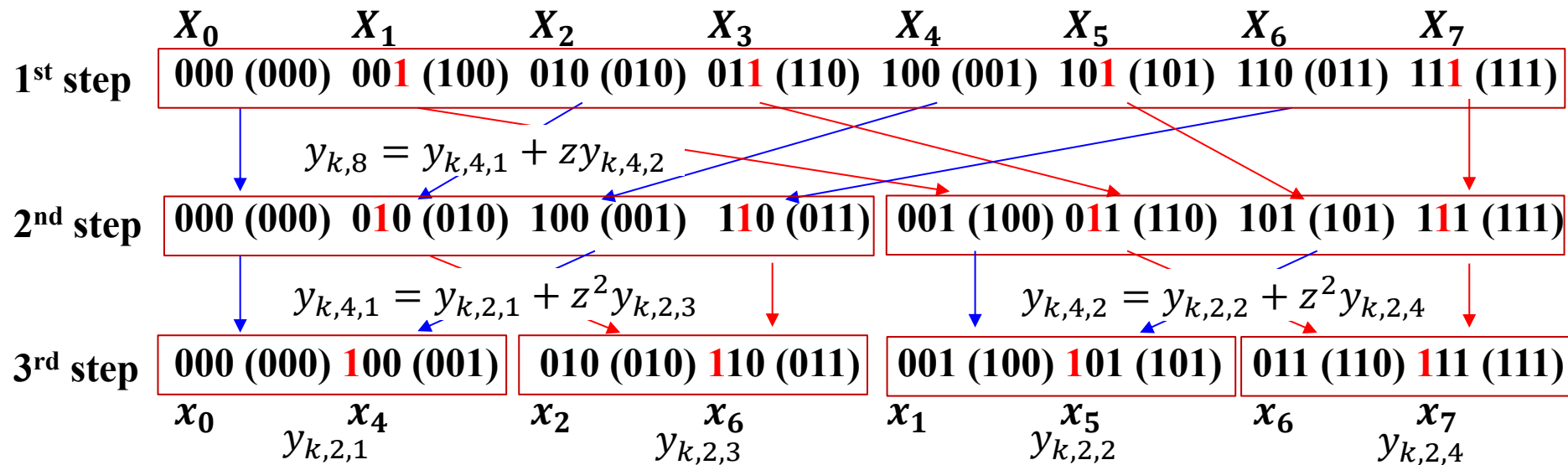
Represent the index number by binary (順序数を二進数であらわす)

At each stage  $k$ , the data are split to two, and **the data of odd order are moved to the second half** (note the order is counted from 0)

$\Rightarrow$  **Data whose  $k$ -th bit is 1 are move to the second half**

$\Rightarrow$  The change of the order numbers corresponds to bit reversal

**Initial data order** (values in parentheses are bitwise reversed)



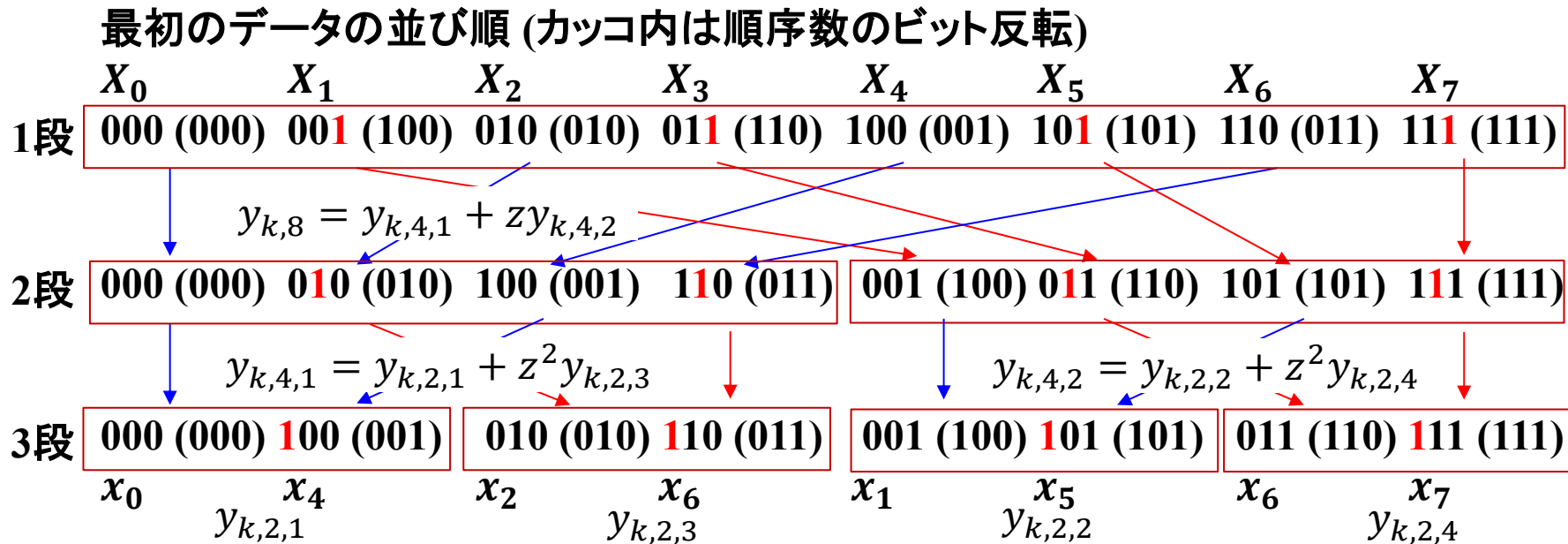
The order to sum up for FFT is different from the order of  $x_i$ .

FFT summation is performed in the order of the bit reversal of the index

# FFT演算の項順序の変換: ビット反転

$N$ 個のデータ列  $x_0x_1x_2 \cdots x_{N-1} \Rightarrow$  FT:  $X_0X_1X_2 \cdots X_{N-1}$   
 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  
 $\Rightarrow$  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」  
 $\Rightarrow$  順序数の変換がビット反転に対応する



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

# Bit reversal (ビット列反転)

**Note:** bit reversal (ビット列反転) != bitwise inversion (ビット反転) ( $\sim x$ , 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<sub>2</sub> を例に

```
ret = 0
```

# ビット反転値を0で初期化

1.  $v0 = val \& 1_2 \Rightarrow 11001_2 \& 001_2 = 1$

# 第1桁のビット値を v0 に保存

2.  $ret = ret | v0 \Rightarrow 0 | 1 = 1_2$

# retの第1桁に v0 を設定

3.  $val = val \>> 1 \Rightarrow 11001_2 \>> 1 = 1100_2$

# 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。

$ret = ret \<< 1 \Rightarrow 1_2 \<< 1 = 10_2$

1.に戻って繰り返す

6.  $v0 = val \& 1_2 \Rightarrow 1100_2 \& 001_2 = 0$

# 第1桁のビット値を v0 に保存

7.  $ret = ret | v0 \Rightarrow 10_2 | 0 = 10_2$

# retの第1桁に v0 を設定

8.  $val = val \>> 1 \Rightarrow 1100_2 \>> 1 = 110_2$

9.  $ret = ret \<< 1 \Rightarrow 10_2 \<< 1 = 100_2$

1.に戻って繰り返す

10.  $v0 = val \& 1_2 \Rightarrow 110_2 \& 001_2 = 0$

11.  $ret = ret | v0 \Rightarrow 100_2 | 0 = 100_2$

12.  $val = val \>> 1 \Rightarrow 110_2 \>> 1 = 11_2$

13.  $ret = ret \<< 1 \Rightarrow 100_2 \<< 1 = 1000_2$

1.に戻って繰り返す

14.  $v0 = val \& 1_2 \Rightarrow 11_2 \& 1_2 = 1$

15.  $ret = ret | v0 \Rightarrow 1000_2 | 1 = 1001_2$

16.  $val = val \>> 1 \Rightarrow 11_2 \>> 1 = 1_2$

17.  $ret = ret \<< 1 \Rightarrow 1001_2 \<< 1 = 10010_2$

1.に戻って繰り返す

18.  $v0 = val \& 1_2 \Rightarrow 1_2 \& 1_2 = 1$

19.  $ret = ret | v0 \Rightarrow 10010_2 | 1 = 10011_2$

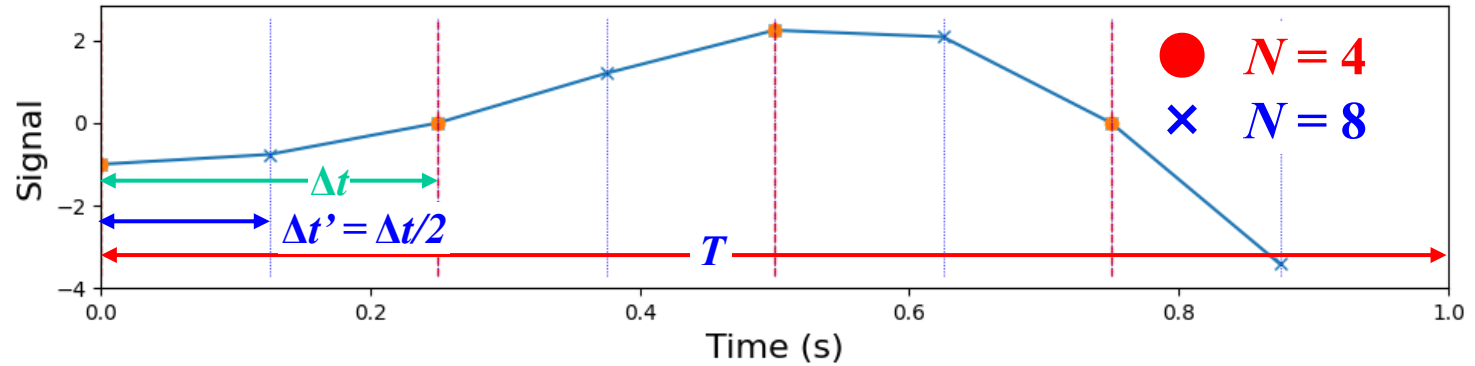
20.  $val = val \>> 1 \Rightarrow 1_2 \>> 1 = 0_2 \Rightarrow$  ループ終了 解:  $ret = 10011_2$

fft.py

# FFT: python numpy.fft.fft()

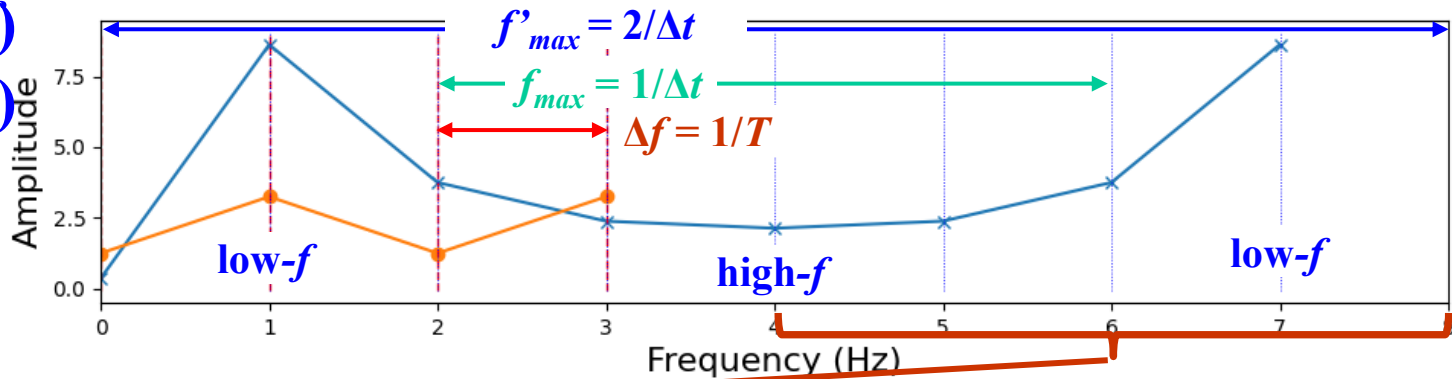
$f(t) = -\cos(2\pi t)(1+5t^2)$ , periodic in  $t = [0, 1)$

$f(t)$   
 $tstep = tmax / N$



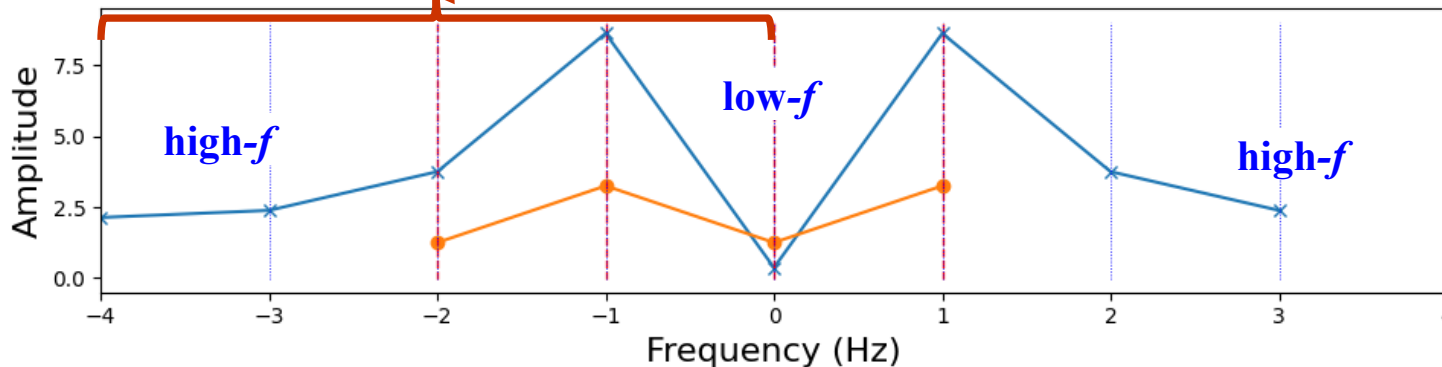
$F(f) = np.fft.fft(f)$

$f = np.fft.fftfreq(N, d=tstep)$



$F = np.fft.fftshift(F)$

$f = np.fft.fftshift(f)$



**NOTE:**

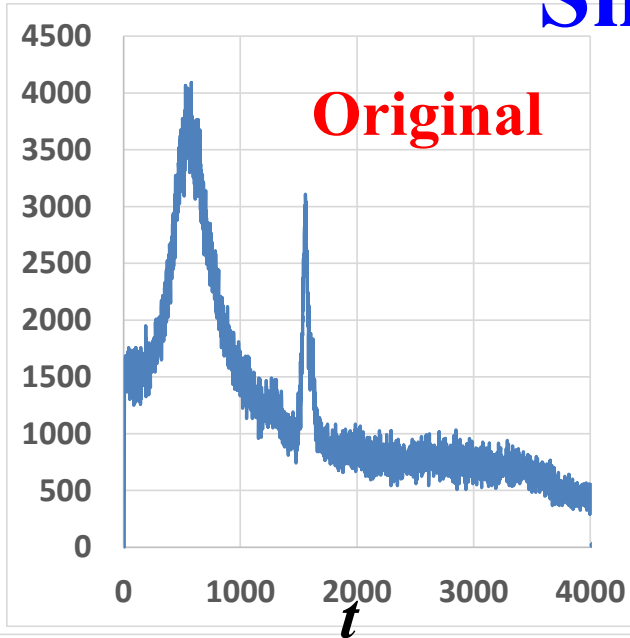
Periodicity of Fourier func:

$$F(f + f_{max}) = F(f)$$

python list:  $a[-n] = a[N - n]$

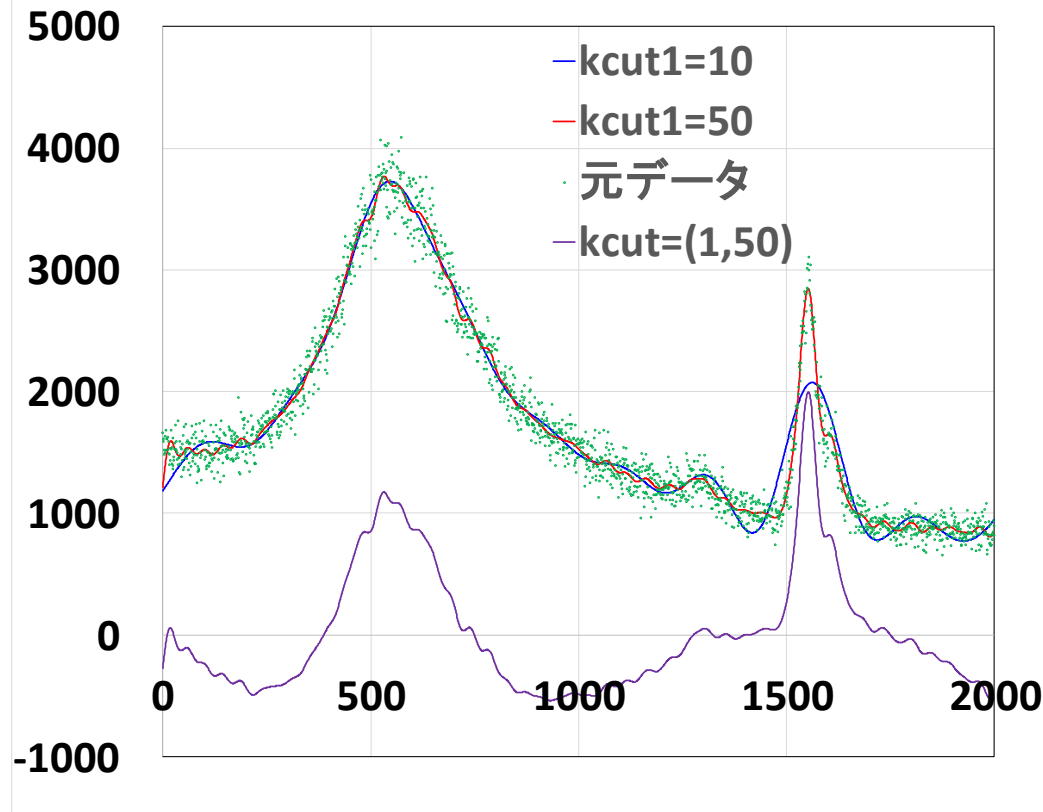
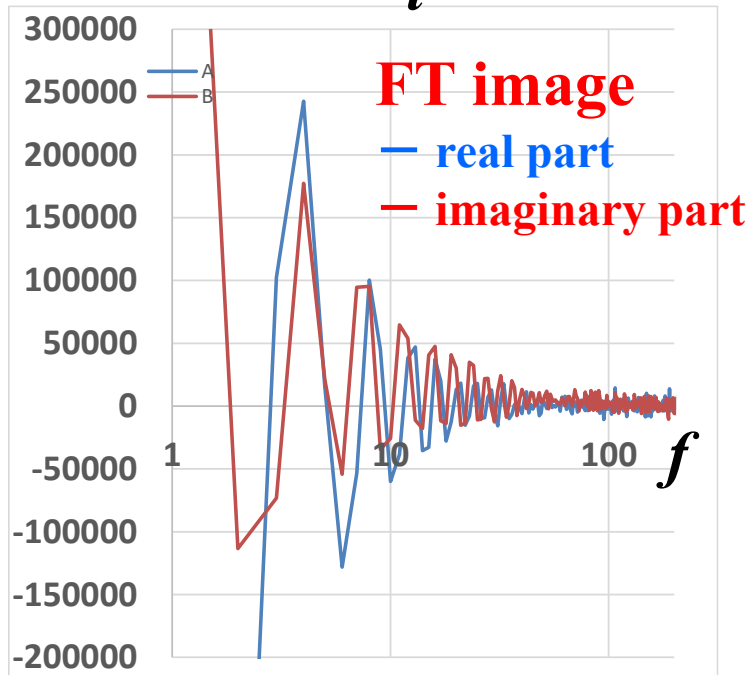
( $N = \text{len}(a)$ )

# Smoothing: FT



Remove high-frequency FT data: Smoothing  
Low-pass filter  
Remove low-frequency FT data: Cut drift  
High-pass filter

Ex. Cut FT data outside  $[k_{\text{cut}0}, k_{\text{cut}1}]$

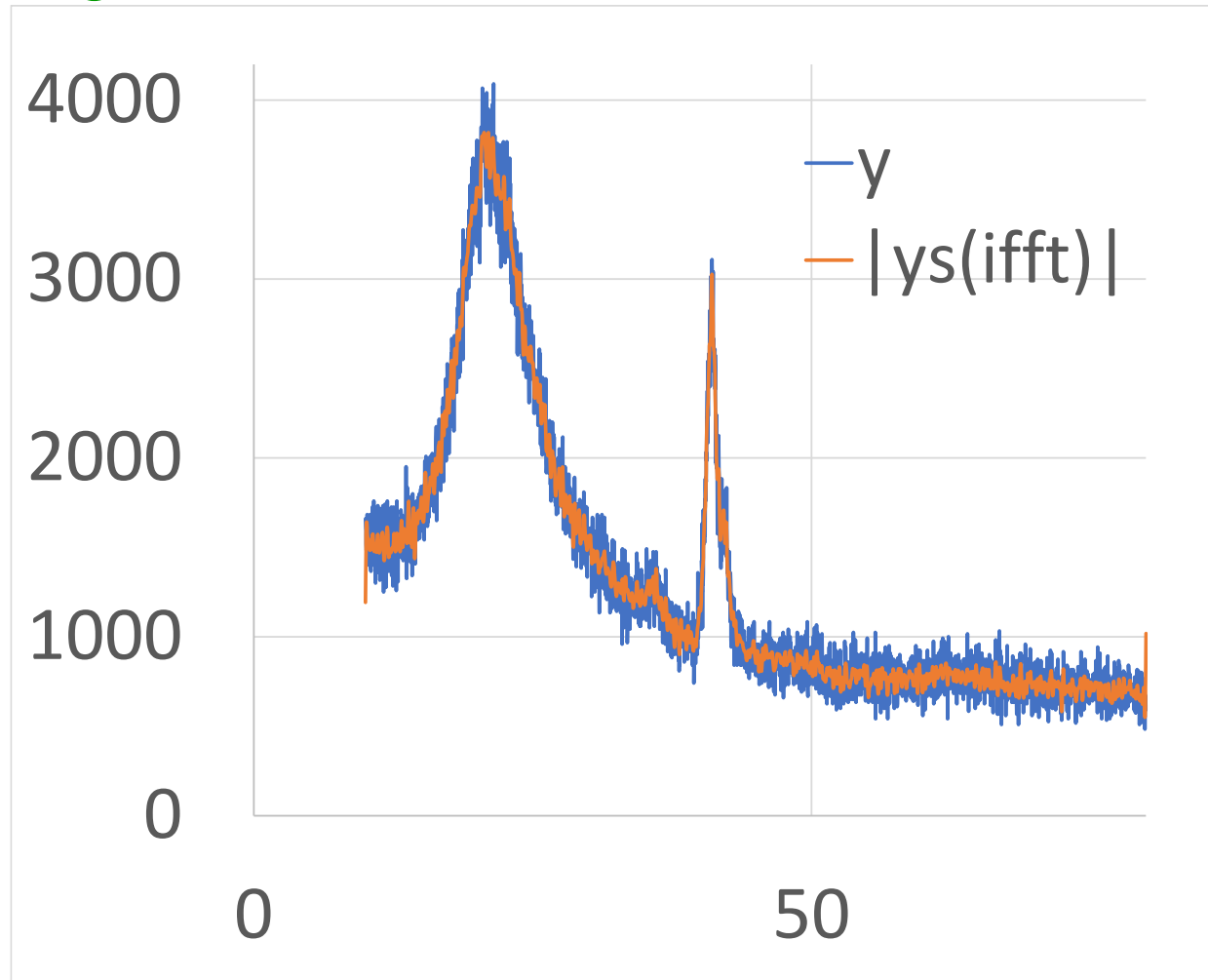


# Program: smoothing-fft.py

Usage: `python smoothing-fft.py xrd.csv 0 5`

(note: the x range is different from the previous slide)

=> `plot smoothing-fft.csv`



# Interpolation by FT

Periodic function  $f(t)$ : Period of  $T$

$N$   $t$  points are given in  $T = N\Delta t$  at uniform step  $\Delta t$ :  $t_j = j\Delta t$  ( $j = 0, \dots, N - 1$ )

can be expanded by  $\exp(i2\pi f_n t_j) = \exp\left(i2\pi \frac{n}{T} t_j\right) = \exp\left(i2\pi \frac{n}{N\Delta t} j\Delta t\right) = \mathbf{\exp\left(i2\pi \frac{nj}{N}\right)}$ ,  $f_n = n/T$

$$\times f(t_j) = \sum_{n=0}^N a_n \exp\left(i2\pi \frac{n}{N} j\right)$$

**For a case for  $N = 4$ :**  $a_j$  are determined so as to reproduce  $f(t_j)$  for **integer  $j$**

$$f(t_j) = a_0 + a_{\pi/4} \mathbf{\exp\left(i \frac{\pi}{4} j\right)} + a_{2\pi/4} \mathbf{\exp\left(i \frac{2\pi}{4} j\right)} + a_{3\pi/4} \mathbf{\exp\left(i \frac{3\pi}{4} j\right)}$$

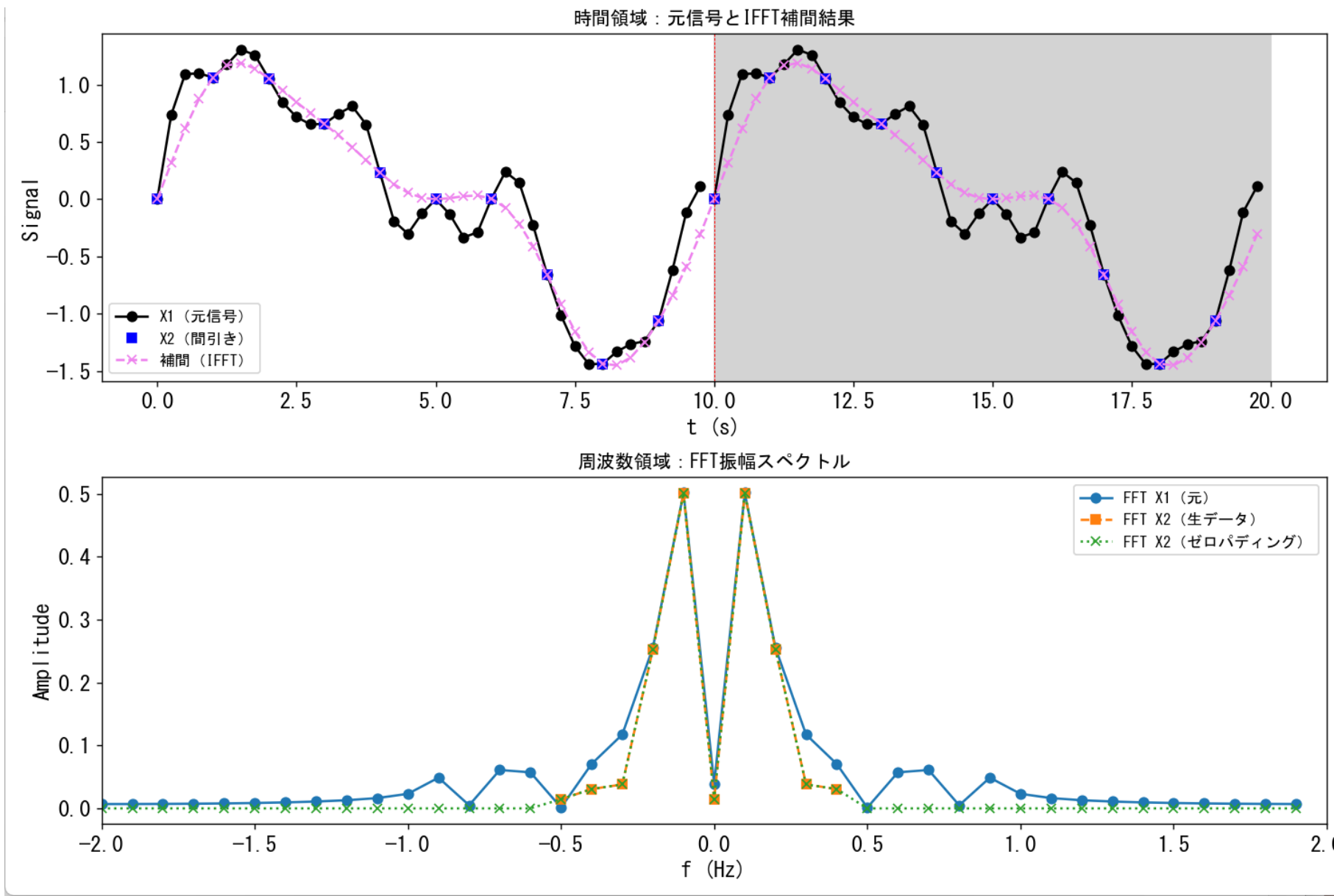
$a_j$  are obtained by Fourier transform

**Interpolate:** for arbitrary **floating-point number**  $t_r = x\Delta t$ :

$$f(t_r = x\Delta t) = a_0 + a_{\pi/4} \mathbf{\exp\left(i \frac{\pi}{4} x\right)} + a_{2\pi/4} \mathbf{\exp\left(i \frac{2\pi}{4} x\right)} + a_{3\pi/4} \mathbf{\exp\left(i \frac{3\pi}{4} x\right)}$$

# Interpolation by FT

> python compare\_FFT.py

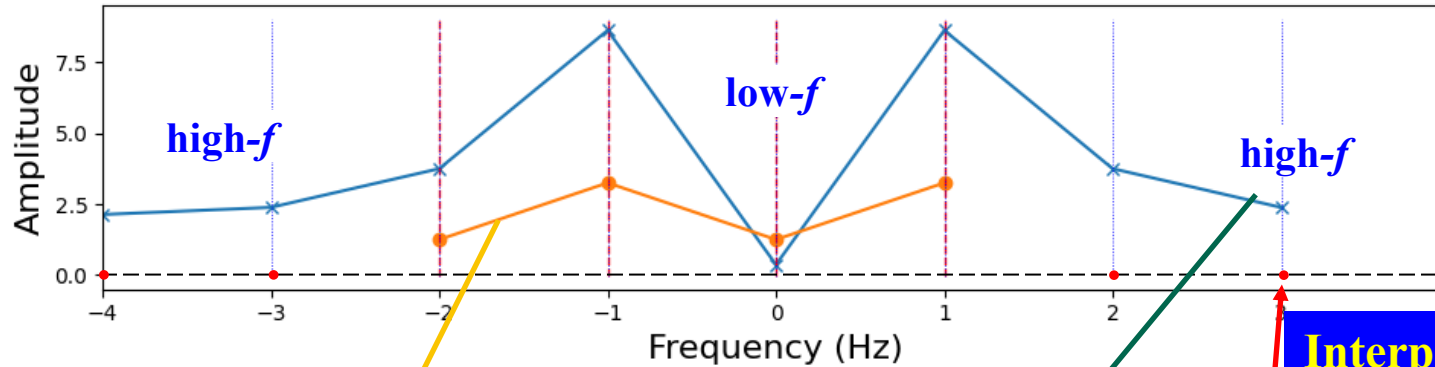


# FFT: python numpy.fft.fft()

fft.py

$f(t) = -\cos(2\pi t)(1+5t^2)$ , periodic in  $t = [0, 1)$

$F(f) = \text{np.fft.fft}(f)$



For a case for  $N = 4$ :

$$f(t_j) = + a_{-2} \exp(-i\pi j) + a_{-1} \exp\left(i\frac{\pi}{2}j\right) + a_0 + a_1 \exp\left(i\frac{\pi}{2}j\right)$$

For a case for  $N = 8$ :

$$f(t'_j) = a'_{-4} \exp(i2\pi j) + a'_{-3} \exp\left(i\frac{3}{2}\pi j\right) + a'_{-2} \exp(i\pi j) + a'_{-1} \exp\left(i\frac{\pi}{2}j\right) \\ + a'_0 + a'_1 \exp\left(i\frac{1}{2}\pi j\right) + a'_2 \exp(i\pi j) + a'_3 \exp\left(i\frac{3}{2}\pi j\right)$$

Interpolation by FFT:

1. Increase number of FTed data
2. Add zeros to additional high-f data
3. Inverse FFT to get interpolated values
4. Correct scale

Interpolate by the FFT result for  $N = 4$ : e.g., to get  $f(t_{1/2} = \Delta t/2)$ :

$$f(t_{1/2}) = a_0 + a_{\pi/4} \exp\left(i\frac{\pi}{8}\right) + a_{2\pi/4} \exp\left(i\frac{2\pi}{8}\right) + a_{3\pi/4} \exp\left(i\frac{6\pi}{8}\right)$$

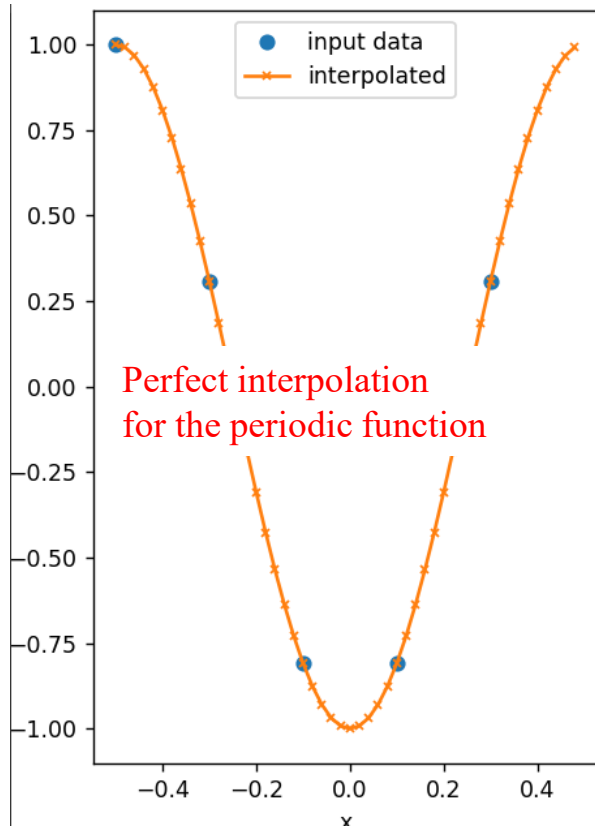
take  $a'_j = a_j$  ( $j = -2, -1, 0, 1$ ),  $a'_j = 0$  (else)

# Interpolation by FFT (Fast Fourier Transform)

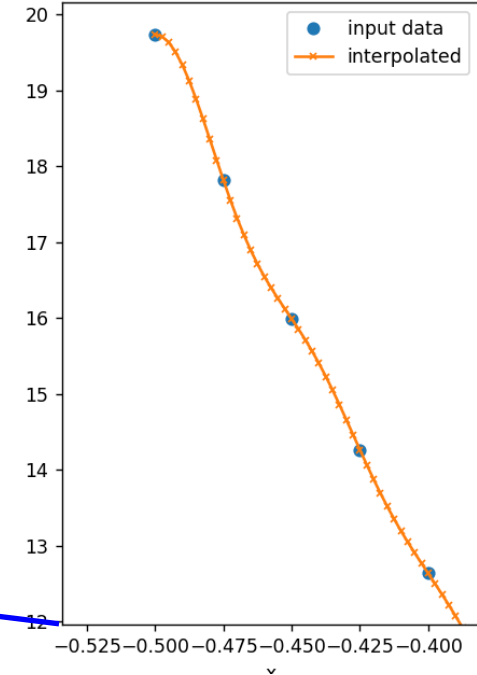
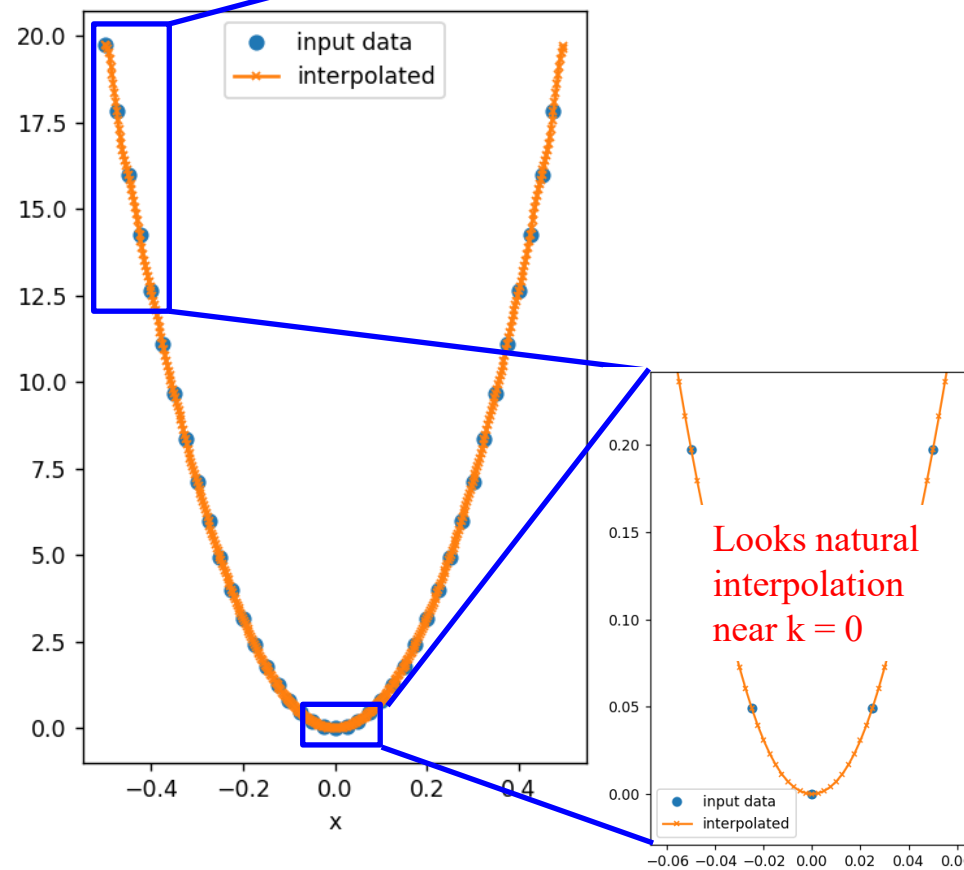
For a periodic function  $E(k)$  (like an energy function  $E(k)$  of crystal), interpolation can be carried out by:

1. Fourier transform  $E(k)$  to  $E^*(x)$
2. Increase the number of data by adding zeros in the high  $|x|$  region (padding)
3. Inverse Fourier transform

> **python interpolate\_fft.py generate**  
data: interpolate\_fft.test.xlsx: TB band



> **python interpolate\_fft.py band\_free\_e.xlsx**  
data: band\_free\_e.xlsx: Free electron band



Interpolated data become a periodic function, producing unnatural deformation near the periodic boundaries

# 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()`

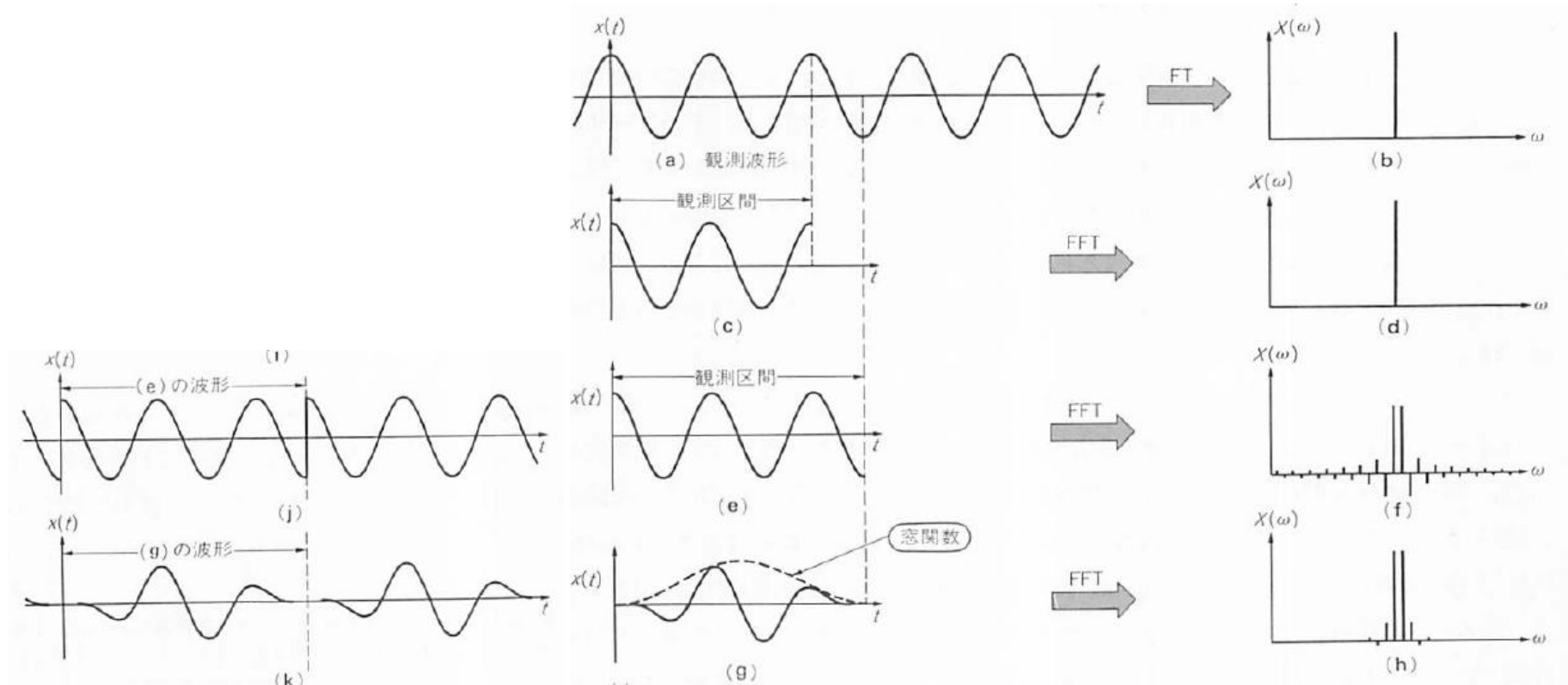
## Time for DFT/FFT (sec)

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  $-\infty$  to  $\infty$ , but DFT/FFT reduces data to finite range  $\Rightarrow$  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  $\Rightarrow$  can be suppressed by **Hann Window** (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)

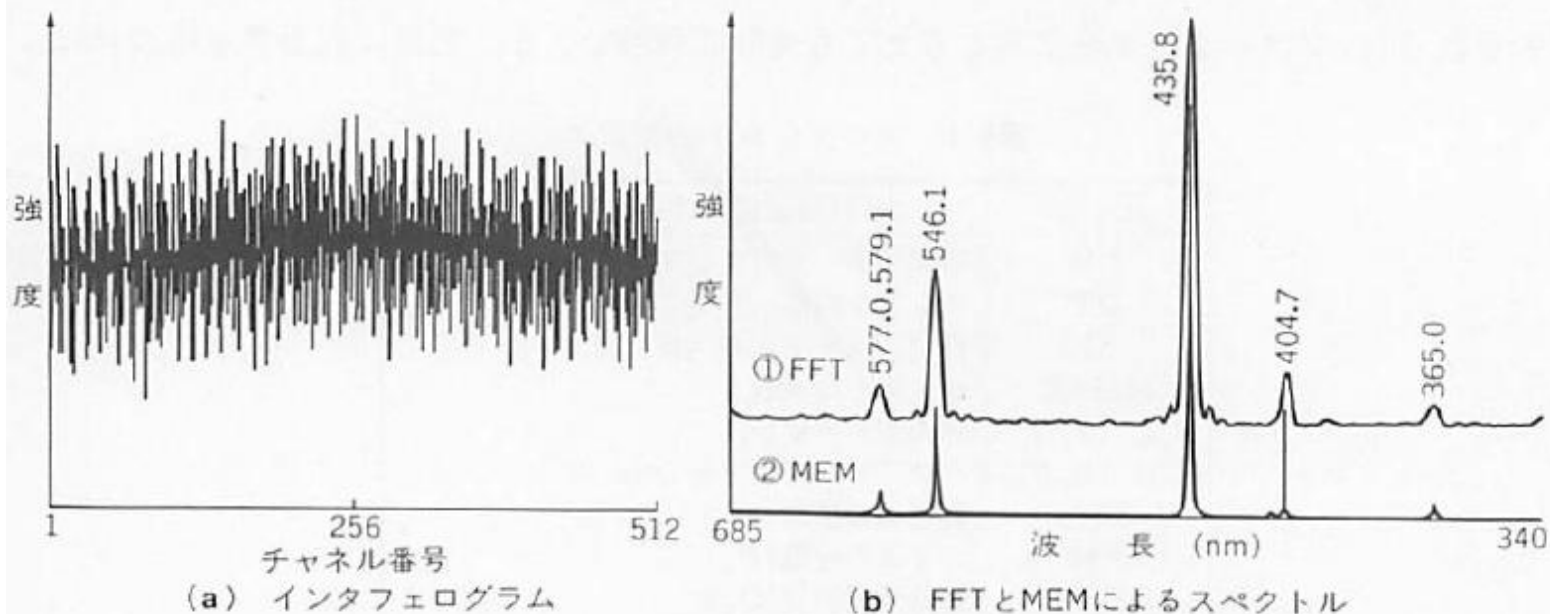
南茂夫, 科学計測のための波形データ処理, CQ出版社 (1986)

## 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) => MEM
  - The order of the autoregressive model  $m$  must be determined
    - So as to minimize Final Prediction Error (最終予測誤差)
- Algorithms: Burg method, etc

## Features

- Sharper spectrum than FFT
- Less ghost peaks and fringes
- But depends on order estimation



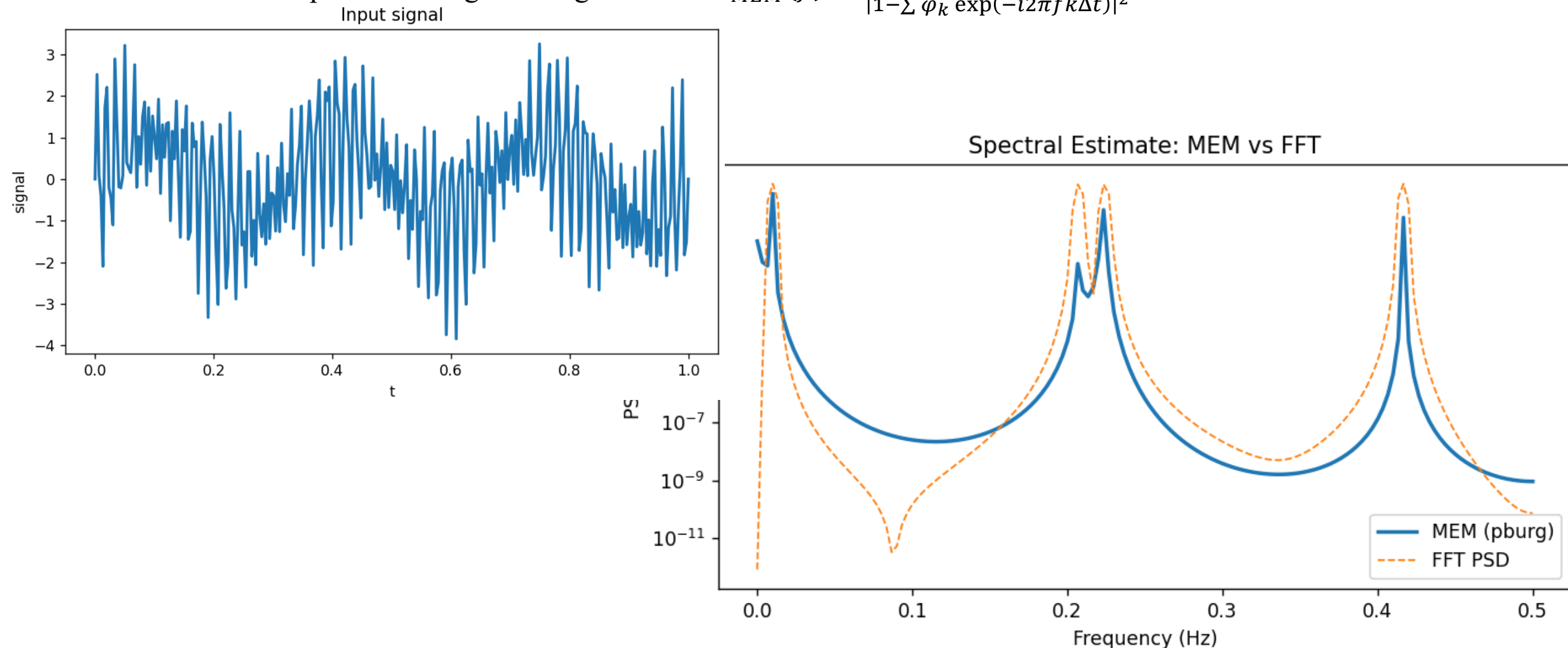
# Program: mem\_spectrum.py

> python mem\_spectrum.py "" 15 log

Order of Autoregression model (AR: 自己回帰モデル): 15

=> Determine regression parameters  $\varphi_k$  ( $k = 1, 2, \dots, order$ ) using past *order* data

+ Estimate MEM spectrum using the Burg method  $S_{MEM}(f) = \frac{2\sigma^2\Delta t}{|1 - \sum \varphi_k \exp(-i2\pi f k \Delta t)|^2}$



# 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  $\rho'$  and suppress fringes and ghost peaks

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

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

=>  $\rho = \exp(\ln \tau + \text{difference Fourier (差フーリエ) term})$

When converged to  $F_{\text{cal}} = F_{\text{obs}}$ ,  $\rho = \tau$  will be achieved

# Schrödinger eq.: **Plane wave method** (平面波法)

**Plane waves are employed as basis set of linear combination**

$$\phi_{\mathbf{k}}(\mathbf{r}) = \exp(i\mathbf{k} \cdot \mathbf{r}) \sum C_{hkl} u_{hkl}(\mathbf{r}) \quad u_{hkl}(\mathbf{r}) = \exp[i\mathbf{G}_{hkl} \cdot \mathbf{r}]$$

Plane waves with wave numbers  $\mathbf{G}_{hkl}$  forms a perfect basis of periodic system

Any function is represented if use all  $G_{hkl}$  for all

=> **In actual calculation, approximate by  $|\mathbf{G}_{hkl}| < G_{\max}$**

$$\begin{vmatrix} H_{11} - ES_{11} & H_{12} - ES_{12} & \cdots & H_{1n} - ES_{1n} \\ H_{21} - ES_{21} & H_{22} - ES_{22} & \cdots & H_{2n} - ES_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n1} - ES_{n1} & H_{n2} - ES_{n2} & \cdots & H_{nn} - ES_{nn} \end{vmatrix} = 0$$

$$\langle u_{h'k'l'} | H | u_{hkl} \rangle = \int e^{-i(\mathbf{k} + \mathbf{G}_{h'k'l'}) \cdot \mathbf{r}} \left[ -\frac{\hbar^2}{2m} \nabla^2 + V(\mathbf{r}) \right] e^{i(\mathbf{k} + \mathbf{G}_{hkl}) \cdot \mathbf{r}} d\mathbf{r}$$

$$= \delta_{hkl, h'k'l'} \frac{\hbar^2}{2m} (\mathbf{k} + \mathbf{G}_{hkl})^2 + \underline{V^*(\mathbf{G}_{hkl} - \mathbf{G}_{h'k'l'})}$$

**Most of PW calculations are done by Fourier transform**

**=> Possibly speed up with GPU**

# Program: 1-D PW method

pw1d.py

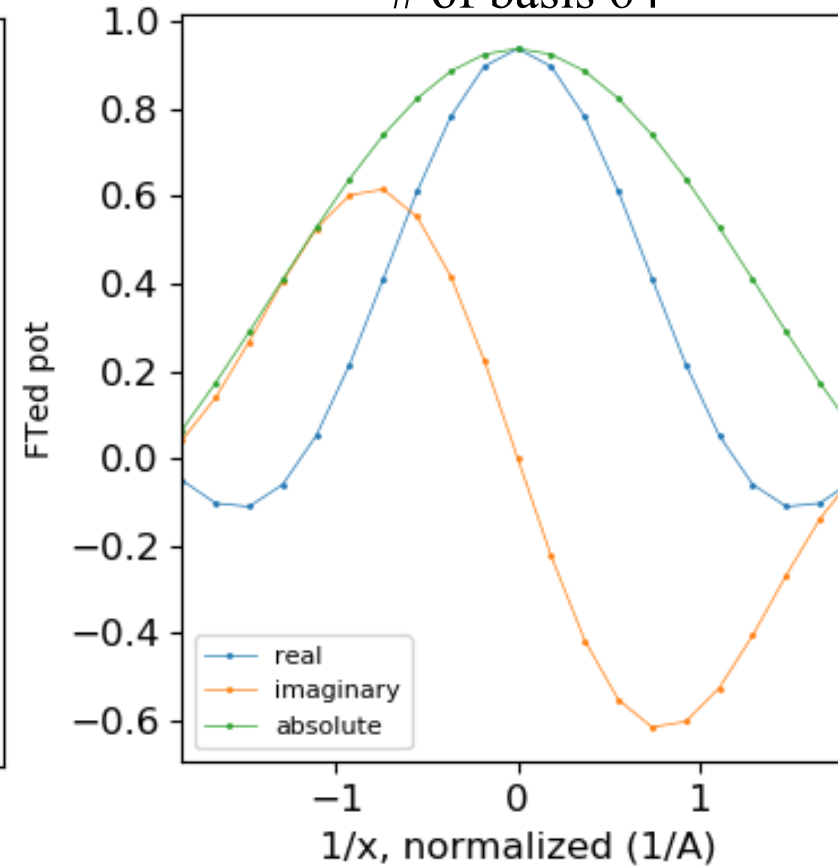
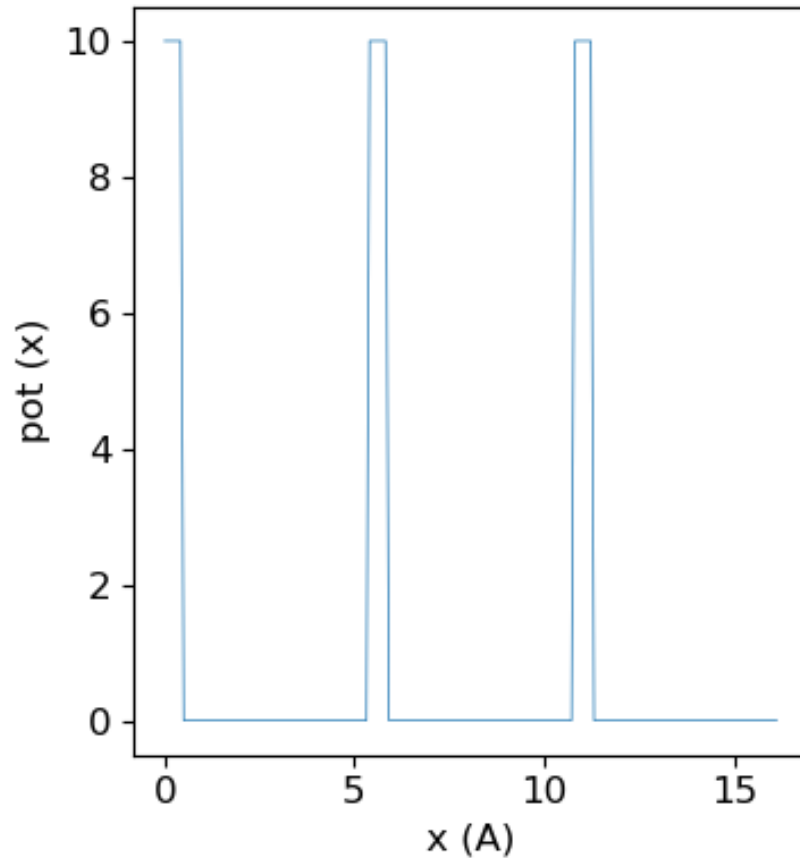
Lattice parameter (Si)  $a = 5.4064 \text{ \AA}$   $m^* = 1.0m_e$

Potential  $V(x)$ : barrier width  $0.5 \text{ \AA}$  barrier height  $10.0 \text{ eV}$

`python pw1d.py ft 5.4064 64 rect 0.5 10.0 9 -0.5 0.5 21`

**FT coefficients of  
potential**

# of basis 64



# Program: 1-D PW method

pw1.py

```
python pw1d.py ft 5.4064 64 rect 0.5 10.0 9 -0.5 0.5 21
```

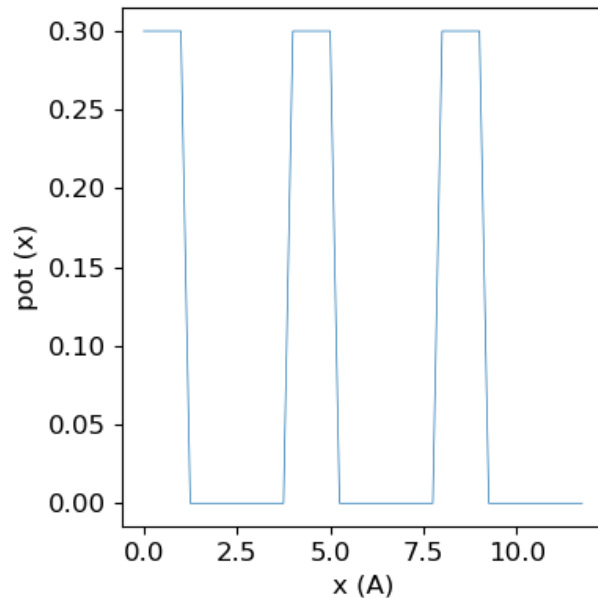
```
python pw1d.py band 5.4064 64 rect 0.5 10.0 3 -0.5 0.5 21
```

```
python pw1d.py wf 5.4064 64 rect 0.5 10.0 3 0.0 0 0.0 16.2192 101
```

$a = 4.0 \text{ \AA}$

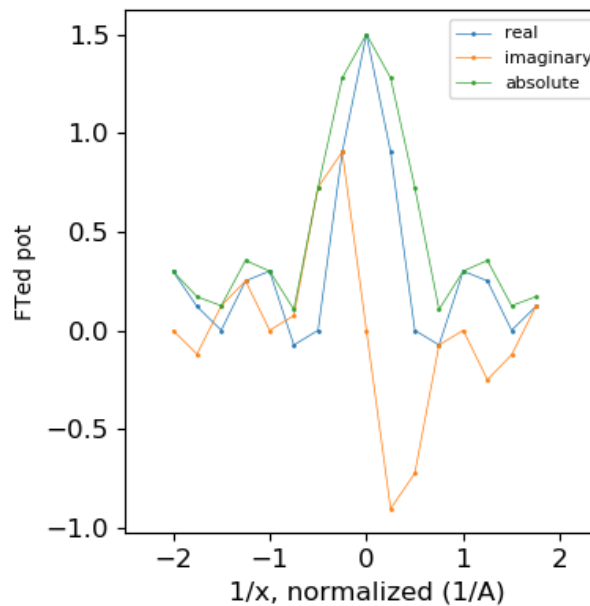
**potential  $V(x)$ :**

$w = 1.0 \text{ \AA}, h = 0.3 \text{ eV}$



**FT coefficients of potential**

# of basis 16



**Band structure**

# of basis 5

