

# Computational Materials Science (計算材料学特論)

## Lecture materials updated

[http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=cms](http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms)



### COMPUTATIONAL MATERIALS SCIENCE 2025 Q2 2025年度Q2 計算材料学特論 (資料: 英語 + 日本語版)

Lecture materials for numerical analyses (by Kamiya)  
数値解析に関する講義資料 (by 河村 貴之 (担当分))

**We will wait for five minutes.**

**In the meantime, please make sure to download the lecture materials**

#### Update News:

- July 1 08:10 Lecture materials on July 1 have been updated ([20250701FFT\\_Matrix.zip](#))

Fourier transformation (フーリエ変換), Matrix (行列), Applications (応用)

Audio explanation generated by Google NotebookLM:

▶ 0:00 / 18:42 — 🔊 ⋮

Google NotebookLMで生成した音声解説:

▶ 0:00 / 6:44 — 🔊 ⋮

- June 27, 20:05 Final version of lecture materials on June 27 have been updated ([20250627EquationOptimize2.zip](#))

Numerical solu

**Also listen the AI-generated brief explanation of the lecture**

Audio explanation generated by Google NotebookLM:

▶ 0:00 / 27:23 — 🔊 ⋮

Google NotebookLMで生成した音声解説:

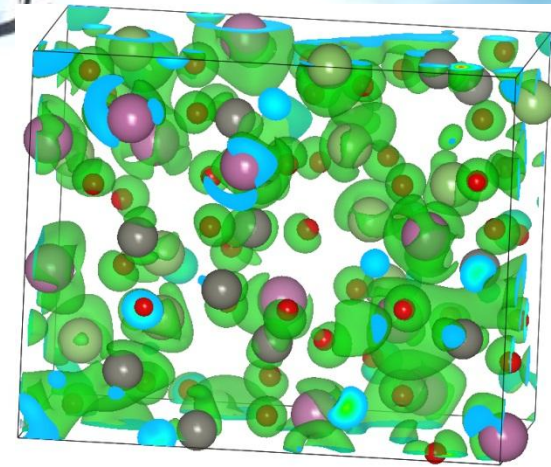
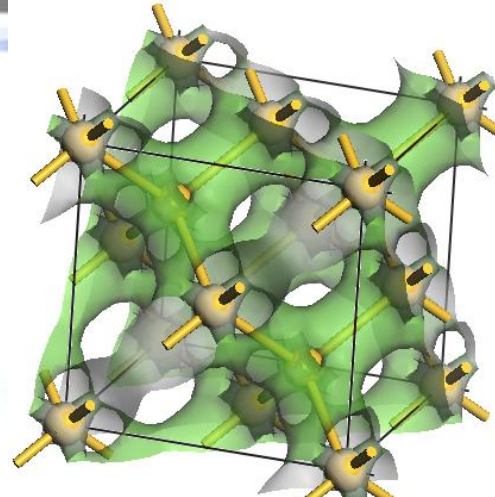
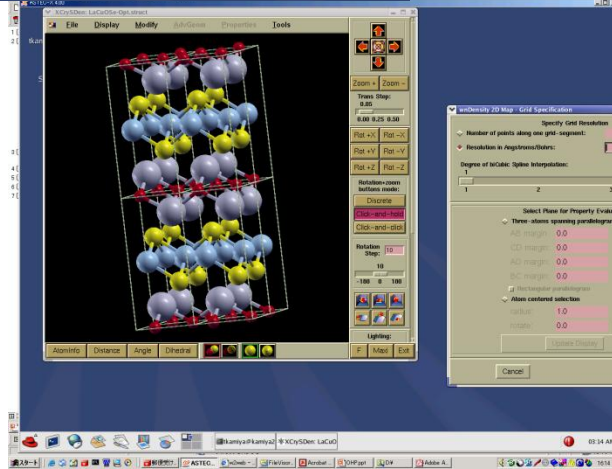
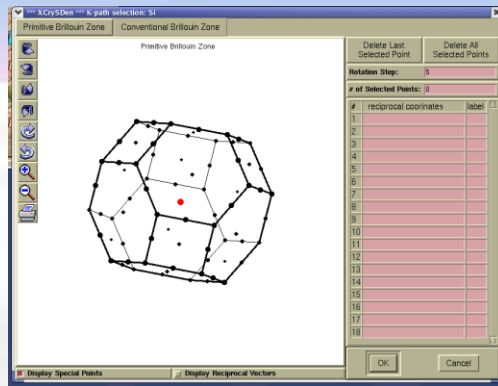
▶ 0:00 / 9:51 — 🔊 ⋮

- June 20, 11:06 Final version of lecture materials on June 20 have been updated ([20250620InterpolateSmoothing2.zip](#))

# Computational Materials Science

## 計算材料学特論

Toshio Kamiya  
神谷利夫



# Class Schedule

Lecture materials (Kamiya's part): [http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=cms](http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms)

- #01 June 10 (Tue) Kamiya (Fundamentals of computer, Sources of errors (コンピュータの基礎、誤差))
- #02 June 13 (Fri) Kamiya (Numerical differentiation/integration (数値微分/積分))
- #03 June 17 (Tue) Kamiya (Differential equation (微分方程式), Molecular dynamics (分子動力学法),  
Interpolation (補間), Smoothing (平滑化))
- #04 June 20 (Fri) Kamiya (Smoothing (平滑化), Linear least-squares method (線形最小二乗法),  
Numerical solutions of equations (方程式の数値解法))
- #05 June 24 (Tue) Canceled
- #06 June 27 (Fri) Kamiya (Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化))
- #07 July 1 (Tue) Kamiya (Fourier transformation (フーリエ変換), Matrix (行列), Applications (応用))
- #08 July 4 (Fri) Sasagawa (Review of quantum theory 1: 量子論おさらい1)
- #09 July 8 (Tue) Sasagawa (Review of quantum theory 2: 量子論おさらい2)
- #10 July 11 (Fri) Sasagawa (First principles calculations: basics 1 第一原理計算: 基礎1)
- #11 July 15 (Tue) Sasagawa (First principles calculations: basics 2 第一原理計算: 基礎2)
- #12 July 18 (Fri) Sasagawa (First principles calc.: applications 1 第一原理計算: 応用1)
- #13 July 22 (Tue) Sasagawa (First principles calc.: applications 2 第一原理計算: 応用2)
- #14 July 25 (Fri) Sasagawa (Classical and Quantum Computers 古典および量子コンピュータ)

# Evaluation (Kamiya)

- **Small quiz**

**Do not evaluate correctness of the answers**

**but consider **how you think** and answer them**

Example of bad case: answering only resulting values in Excel

Good case: Answer with Excel equations / python program etc  
so that I can check how you solve the assignments

- **Term-end assignment**

**Problems will be given at the end of Q2 from LMS**

# **Explanation of the answers**

**課題解答の解説**

# PROBLEM, June 27

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

## PROBLEM:

Solve  $5\cos(x) - x = 0$ .

- Plot the functions  $y = 5\cos(x)$  and  $y = x$  in the range  $x = 0 - 3$ , find an initial  $x$  for Newton-Raphson method.
- Solve  $5\cos(x) - x = 0$  by Newton-Raphson method at least with four significant digits.
- Mandatory:** Tell your opinions or suggestions for improvements regarding how the lectures are conducted, particularly in relation to the use of AI tools like NotebookLM.
- Optional:** Propose if you have any other numerical analysis you want to learn in Computational Materials Science
- Optional:** Propose if you have any python program (should be simple) you want to learn



# PROBLEM, June 27

See [equation\\_answer.xlsx](#)

## **PROBLEM, June 27**

**Q: how the lectures are conducted, the use of AI tools like NotebookLM.  
other numerical analysis, python programs**

**A:**

- Regarding the use of AI, some students concern about its usage and verifying the accuracy of responses, but the overall opinions are positive.
- There are apprehensions about hallucination (incorrect outputs).
- A student suggest to convert the NotebookLM share link into a QR code and to embed it in slides.



# PROBLEM, June 27

## Additional Suggestions:

- Increase the number of concrete exercises
- Provide project-based assignments like simple data analysis or creating models with AI

A: I'll try but in the given time slot of CMS

- Explain important parts in Japanese

A: I did it before but bilingual lecture took too long time  
even if the Japanese parts are short

- Start classes with an introduction to python environment setup and AI usage

A: As I explained at the beginning of the lecture, python is better to know  
but not mandatory for CMS

# PROBLEM, June 27

Q: How to use AI (prompting), data analysis using AI?

Japanese Tutorial page:

[http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=tutorial](http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=tutorial)

## 材料計算科学・データ解析チュートリアルコース

▶ 第4回 「生成AIの利用例: 一般的な質問と画像生成 (Part 1: 一般的な質問)」

Google NotebookLMで生成した音声解説: ▶ 0:00 / 6:39 — 🔊 ⋮

▶ 第4回 「生成AIの利用例: 一般的な質問と画像生成 (Part 2: データの整形、会議資料の作成)」

▶ 第4回 「生成AIの利用例: 一般的な質問と画像生成 (Part 3: 数学の公式の証明)」

▶ 第4回 「生成AIの利用例: 一般的な質問と画像生成 (Part 4: 画像生成)」

▶ 第5回 「生成AIの利用例: 翻訳 (翻訳の注意、比較、クロスチェック)」

Google NotebookLMで生成した音声解説: ▶ 0:00 / 7:01 — 🔊 ⋮

▶ 第6回 「生成AIの利用例: 良いプログラムに必要な機能とtemplateを用いた高機能プログラムの作成」

Google NotebookLMで生成した音声解説: ▶ 0:00 / 7:06 — 🔊 ⋮

▶ 第6回 「生成AIの利用例: 既存プログラムの理解と改良」

Google NotebookLMで生成した音声解説: ▶ 0:00 / 8:38 — 🔊 ⋮

# PROBLEM, June 27

Q: How to use AI (prompting), data analysis using AI?

学部2年 材料計算科学基礎

[http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=fcms](http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=fcms)

[D<sup>2</sup>MatE Top](#)  
[source HTML](#)  
[display HTML](#)  
  
**2024年度Q4 材料計算科学基礎**  
  
**python**  
[python文法](#)  
[プログラム例](#)  
[仮想環境](#)  
[ライブラリのインストール](#)  
  
**生成AI**  
[講義における生成AI](#)

[source HTML](#)[display HTML](#)

[D2MatE top](#) [計算材料科学基礎 top](#)  
  

## 講義で使ったLLM: 2024年度Q4 材料計算科学基礎

- **状況:** 非線形最小二乗法のプログラムを作成した  
**プロンプト:** pythonのcurve\_fit()を使って、非線形最小二乗法を行うプログラムを作成してください  
**回答:** 以下は、回答プログラムを修正したもの  
▶ 02/nlsq\_curvefit.py ([download/show](#))
- **状況:** 試作プログラムを汎用的に使えるようにするため、ハードコーディング部分を外部入力(起動時引数、入力ファイルの内容)で設定できるように修正する  
**プロンプト:** 以下のプログラムに次の修正をしてください  
#グラフで日本語が表示できるように  
#入力ファイル名をargvでうけとれるように  
#初期値p0はプログラムの最初の方で定義し、argvでも受け取れるようにする  
#Excelのラベル文字列決め打ちではなく、1列目をx、2列目をyに入れる  
#グラフのxlabelは、Excelの1列目のラベル、ylabelは2列目のラベル  
#titleは入力ファイル名  
▶ 03/実験.py ([download/show](#))

# PROBLEM, June 27

**Q: Diagonalization and band calculations**

**A:** See `benezene.py` (Huckel approximation) and `pw1d.py` (plane wave band structure)

**Q: python virtual databases**

**A:** (i) Easiest way: Use a **dictionary** type variable as a **simple Key – Value database**

(ii) **JSON** type variable (equivalent to a tree-structured dictionary)  
can handle with a more complex database

(iii) **pandas.DataFrame** can be used as a simple database  
with **search/extraction functions**.

Further, you may use **pandasql** to perform **SQL** commands.

(iii) Use **SQL server** with a memory mode.

Example for **SQLite3**: `connection = sqlite3.connect(':memory:')`

# PROBLEM, June 27

**Q: Finite Element Method (FEM)**

**A:** It is important but difficult to be covered by CMS  
as FEM should include some complex steps:

- (i) Generate mesh
- (ii) Construct equations
- (iii) Solve multi-dimensional simultaneous differential equations  
with respect to  $\mathbf{r}$  and  $t$

# PROBLEM, June 27

**Q: Machine learning methods to predict a future variation**

**Japanese Tutorial page:**

**[http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=tutorial](http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=tutorial)**

source HTML display HTML

## 材料計算科学・データ解析チュートリアルコース

文部科学省「データ創出・活用型マテリアル研究開発プロジェクト」半導体拠点 D<sup>2</sup>MatE では、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を開催いたします。

- ▶ 2022年度第1回 「PHYSBOを利用した強化学習プログラムの使い方」
- ▶ 2022年度第2回 「線形最小二乗法、回帰、最適化の基礎」
- ▶ 2022年度第3回 「Ridge回帰、機械学習」
- ▶ 2022年度第4回 「ガウス過程とベイズ最適化、非線形最適化・最小二乗法」
- ▶ 2022年度第5回 「非線形最小二乗法、スペクトル解析」

## **PROBLEM, June 27**

**Q: Machine learning methods to predict a future variation**

**A:** Unlike least-squares method, ML does not need to suppose a function shape for regression.



# 普段使っている最小二乗法と機械学習回帰の違い

## 普段使っている最小二乗法:

- ・モデルが物理・化学に基づく数式で表されている
- ・モデル変数は少ない: 化学組成  $x$  と温度  $T$  など

モデル (Arrhenius plot):  $\log \sigma(T, x) = a_0(x) + a_1(x) \frac{1}{T}$   
 $= a_{0,0} + a_{0,x} x + (a_{1,0} + a_{1,x} x) \frac{1}{T}$

## パラメトリックモデル

## 機械学習回帰:

- ・良いフィッティング結果が得られれば、モデルの根拠は問わない  
モデルの関数は非常に柔軟
- ・モデル変数(記述子)が多い場合を扱うことも多い:  
試料の作製条件 (反応温度、反応時間、酸素分圧、組成・・・)  
モデル:  $\sigma(T) = a_0 + a_1 T + a_2 t + a_3 P + \dots$

## ノンパラメトリックモデル

# Non-parameteric regression

線形回帰でも

基底関数の線形結合を使うと

任意関数を表すことができる

(基底関数が対称性・正定値性を満たすKernel関数の場合、Kernel回帰と同じ)

例: ガウス基底関数  $g_1(x) = \exp \left[ - \left( \frac{(x - x_{0,i})^2}{2\sigma^2} \right) \right]$

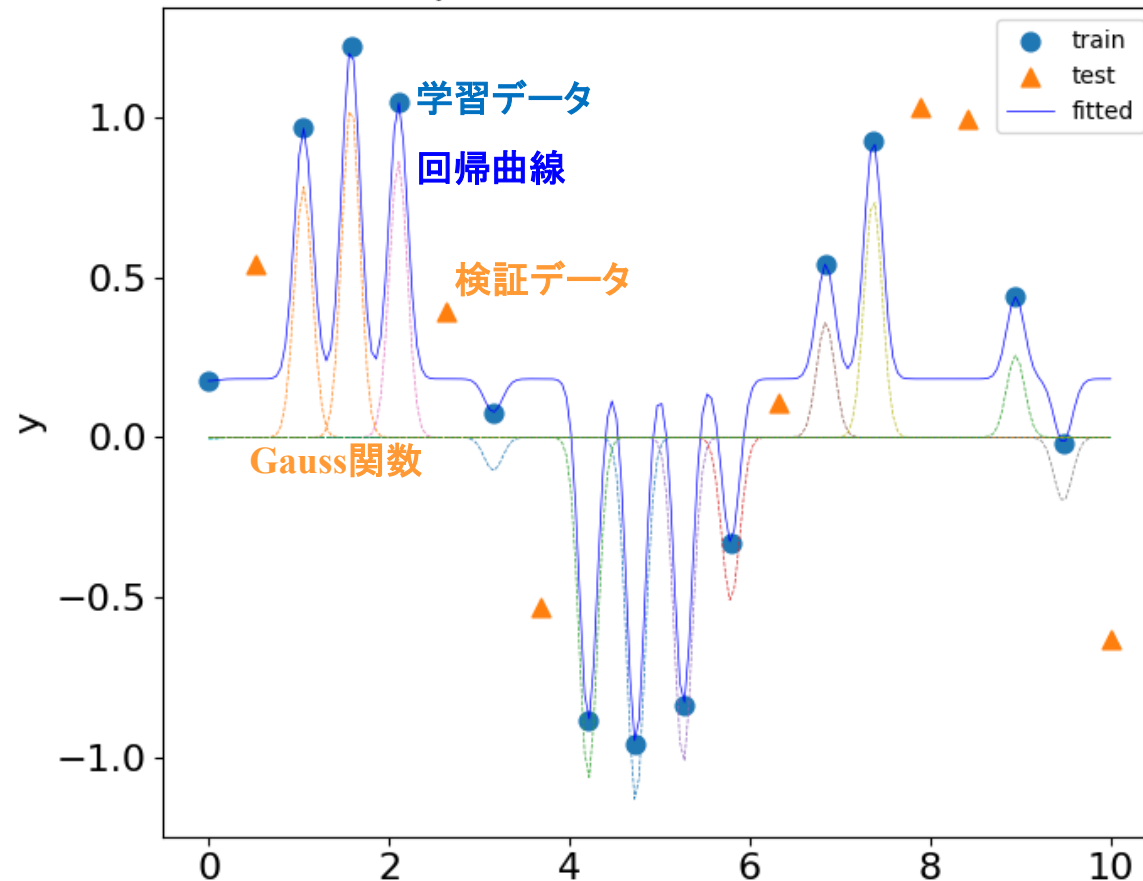
モデル:  $f(x) = a_1 g_1(x^{(1)}) + a_2 g_1(x^{(2)}) + \dots = \sum_{k=0}^p a_k g_1(x^{(k)})$

$g_i(x)$ の数を多くし、隣接する $g_i(x)$ が重なるようにすると、

柔軟に任意の関数を再現できる: ノンパラメトリック回帰

# Kernel ridge回帰

- **ノンパラメトリック回帰**: 多くのパラメータをもつ柔軟なモデル関数で回帰。  
具体的な関数形を知らなくても回帰モデルを作れる。
- **Kernel関数**: ここでは、一般関数の最小二乗法ででてきた基底関数  $f_i(x)$  と同じと考えていい  
学習データ点  $x_i$  毎に、 $x_i$  を中心とし、幅  $w$  の **Gauss関数**  $G(x; x_i, w)$  を置く
- **Ridge回帰**:  
損失関数として残差二乗和  $\sum_{j=1}^n (\sum_{k=1}^p a_k x^{(k)}_j - y_j)^2$  の他に正則化項  $\alpha \sum_{k=0}^p a_k^2$  を加えて**過学習を抑制**する



# 過学習している例

> `python gaussian_ridge.py 0.7 0.1 0.001` (Gauss基底関数の幅が0.1)

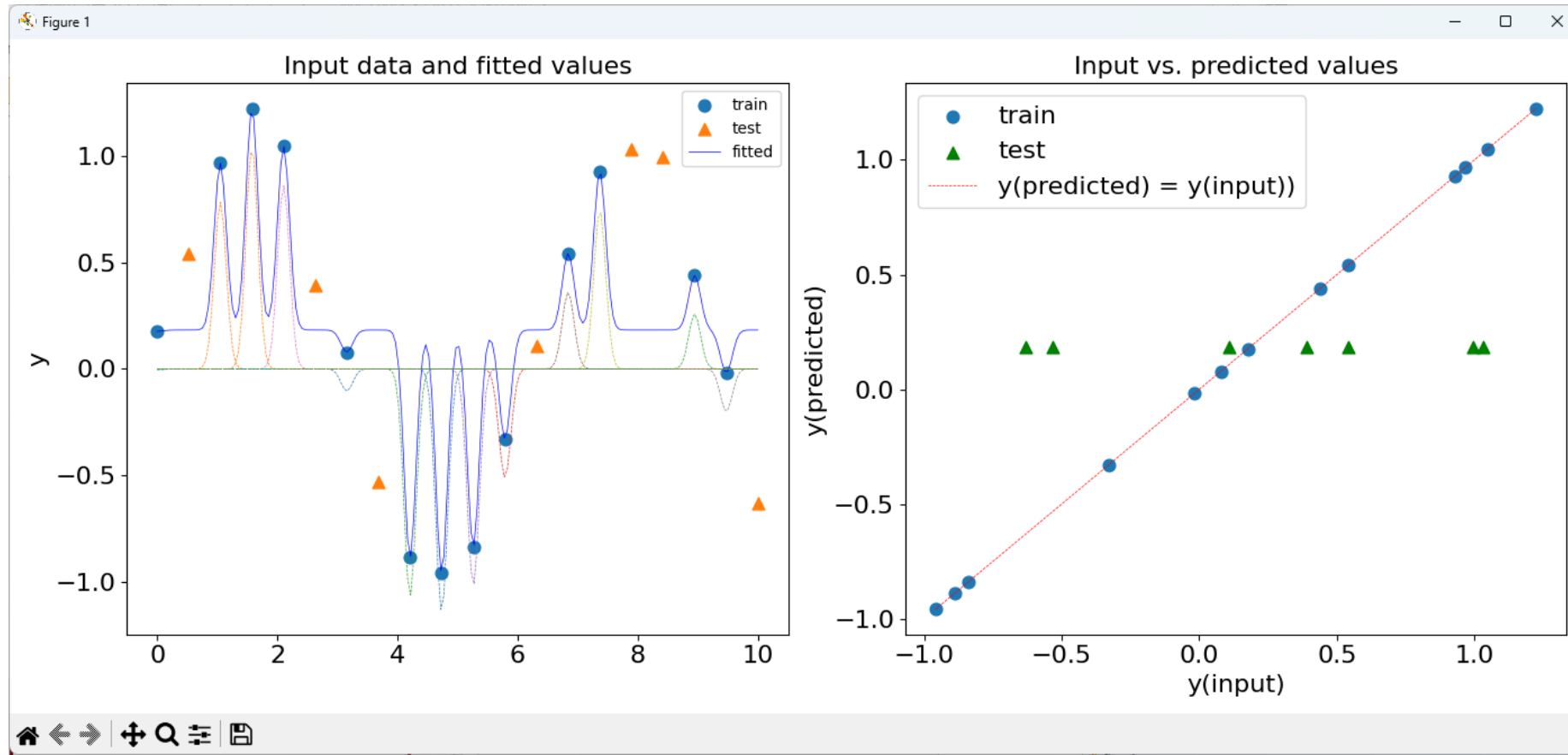
`ratio_train = 0.7`

`w = 0.1`

`alpha = 0.001`

Train MAE: 0.000622842, RMSE: 0.000732994, R2: 0.999999

Test MAE : 0.546841, RMSE: 0.624088, R2: -0.021084



# 基底関数 (幅 $w$ ) の選択で過学習を抑える

> `python gaussian_ridge.py 0.7 0.5 0.001` (Gauss基底関数の幅が0.5)

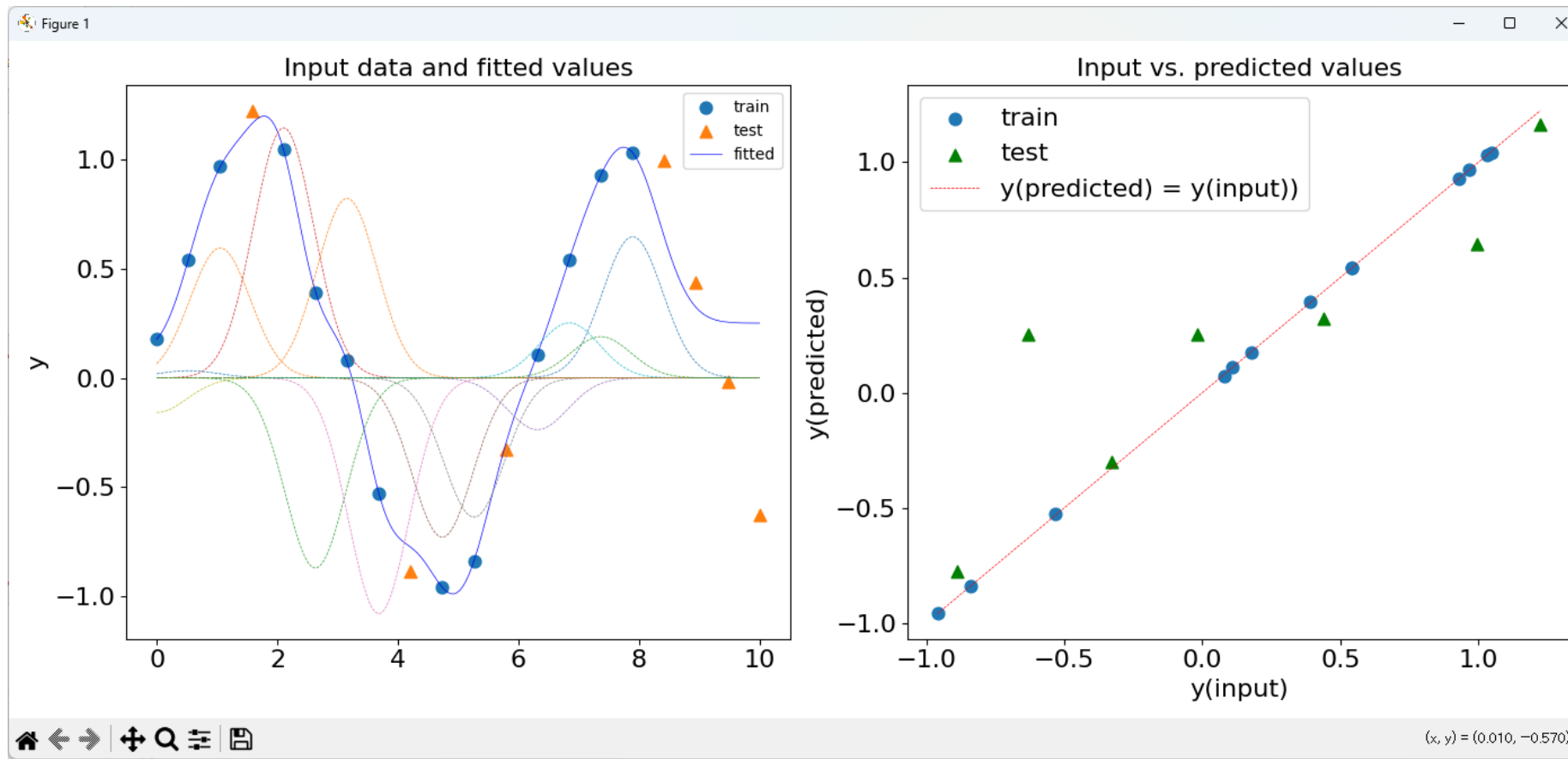
`ratio_train = 0.7`

`w = 0.5`

`alpha = 0.001`

Train MAE: 0.00203944, RMSE: 0.0031156, R2: 0.999978

Test MAE : 0.259566, RMSE: 0.378193, R2: 0.741828



## PROBLEM, June 27

**Q: Optimization, particularly global optimization methods**

**A:** Simplex method has a good larger-size global optimization, but usually the following methods would be used:

- Genetic algorithm (GA, 遺伝的アルゴリズム)
- Particle swarm optimization (PSO, 粒子群最適化)
- Replica exchange MC (REMC, レプリカ交換モンテカルロ法)

See `ga_optimization.py`, `optimize_ga_swarm_remc.py`

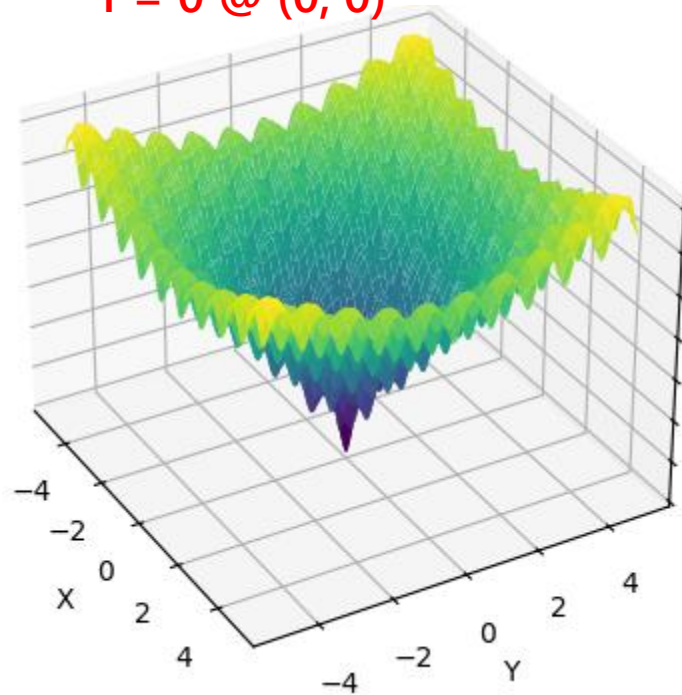
but those programs should be heavily improved for practical use

# Comparison

Python optimize\_ga\_swarm\_remc.py: simplex, ga, pso, remc

Minimize Ackley function:

$f = 0$  @ (0, 0)



Initial guess range [-30, 30] / simplex: (5,5)~[-30,30]

# of T (remc): 10

Tol:  $10^{-3}$ , nmaxiter: 1000

		GA	PSO	REMC	Simplex
nparents	5				
fmin		6.213	<b>0.00042</b>	2.761	<b>9.46E-05</b>
icall		9002	235	24001	82
iter		<b>1000</b>	28	<b>1000</b>	40
icall/iter/nparents		1.8004	1.678571	4.8002	2.05
nparents	10				
fmin		1.867	<b>0.0006</b>	0.58908	<b>9.46E-05</b>
icall		21001	380	49001	82
iter		<b>1000</b>	35	<b>1000</b>	40
icall/iter/nparents		2.1001	1.085714	4.9001	2.05



## **PROBLEM, June 27**

**Q: Predicting future changes from parameter variations through simplified machine learning**

**Q: Programs directly tied to material development**

**A:** You can search these subjects by

Materials Informatics / Materials Data transformation.

Python library pymatgen & DB Materials Project etc would help

<https://next-gen.materialsproject.org/>

# PROBLEM, June 27

## Example python programs to download material data

[http://conf.mdxes.iir.isct.ac.jp/D2MatE/D2MatE\\_programs.html?page=tiny](http://conf.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=tiny)

### Structure database

- ▶ Materials Projectから検索し、CIFファイルを保存: [get\\_cif.py \(download/show\)](#)

必要ライブラリ: pymatgen

使用方法: `python get_cif.py formula output_format prec`

例: `python get_cif.py BaTiO3`

- ▶ Materials Projectから検索し、電子バンド構造、状態密度を保存: [get\\_band\\_dos.py \(download/show\)](#)

必要ライブラリ: pandas, matplotlib, pymatgen

使用方法: `python get_band_dos.py formula k_path`

k\_path: Hinuma | Latimer-Munro | uniform | Setyawan-Curtarolo

例: `python get_band_dos.py BaTiO3`

- ▶ Materials Projectから検索し、フォノンバンド構造、状態密度を保存: [get\\_phonon.py \(download/show\)](#)

必要ライブラリ: pandas, matplotlib, pymatgen

使用方法: `python get_phase_diagram_dos.py chemsys`

例: `python get_phase_diagram.py Li-Fe-O`

- ▶ Materials Projectから検索し、状態図を保存: [get\\_phase\\_diagram.py \(download/show\)](#)

必要ライブラリ: pandas, matplotlib, pymatgen

使用方法: `python get_phase_diagram_dos.py chemsys`

例: `python get_phase_diagram.py Li-Fe-O`

**How to solve equations?**

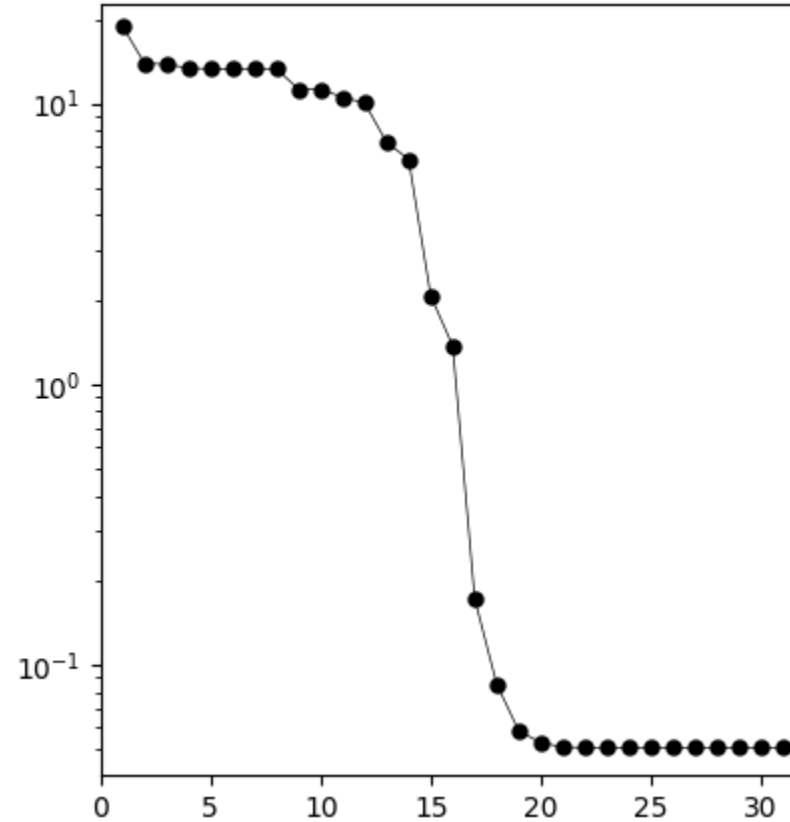
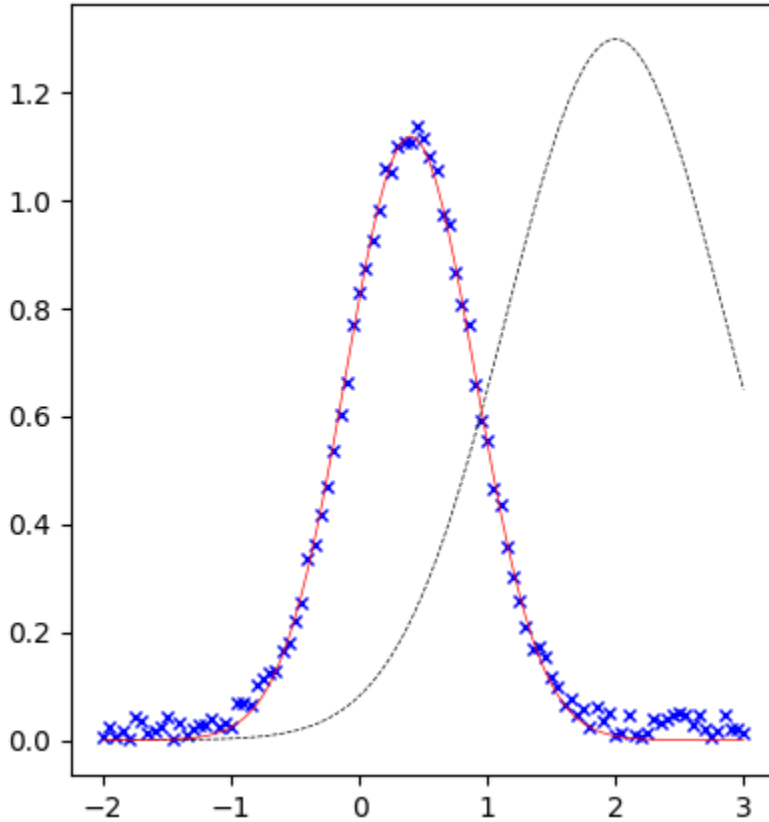
**More sophisticated algorithms**

# Converged

python peakfit.py fit peak.xlsx cg 1.3 2.0 1.0

method: cg

initial values:  $I_0 = 1.3$ ,  $x_0 = 2.0$ ,  $w = 1.0$



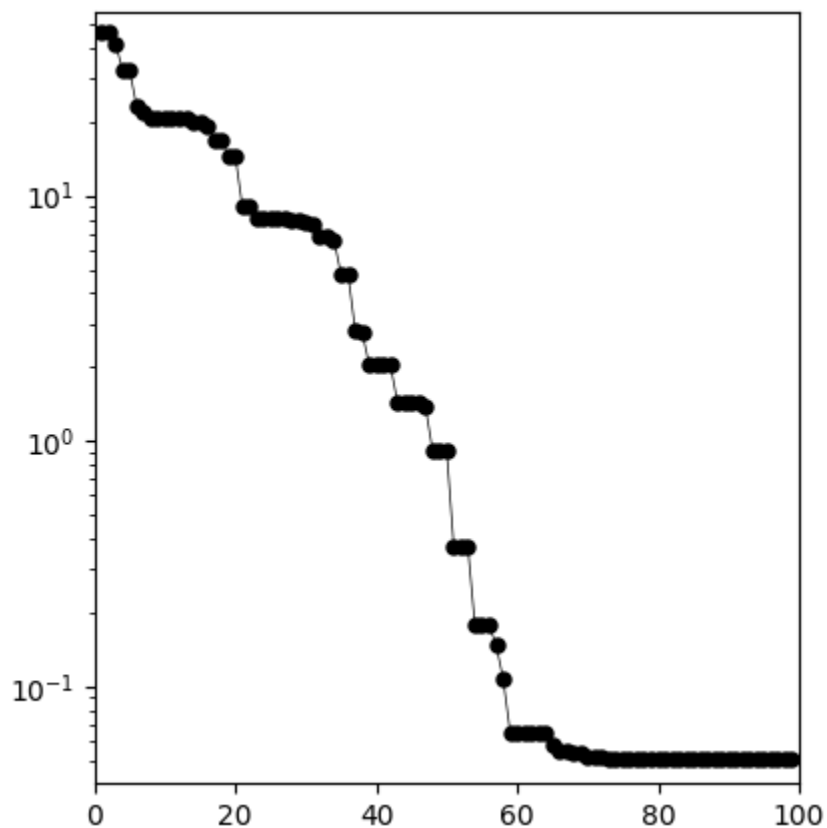
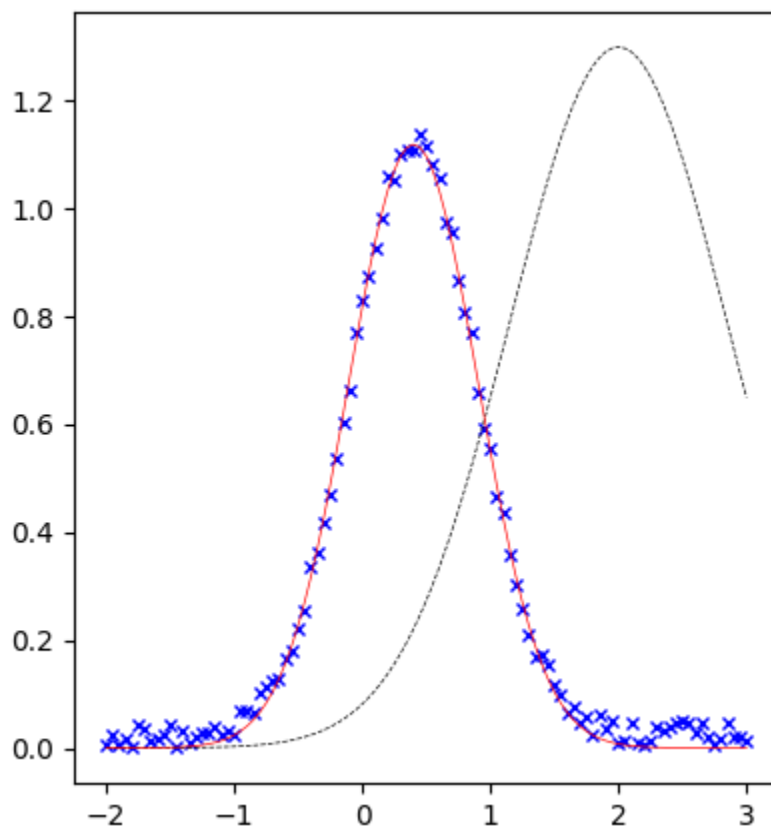
The result may be converged if  $w$  is wide enough to cover some region of the curve, even if the initial peak position is out of the FWHM of the target peak

# Converged

python peakfit.py fit peak.xlsx nelder-mead 1.3 2.0 1.0

method: nelder-mead (SIMPLEX)

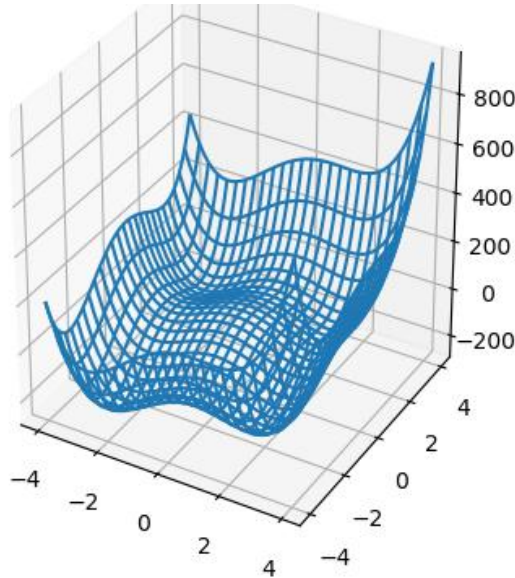
initial values:  $I_0 = 1.3$ ,  $x_0 = 2.0$ ,  $w = 1.0$



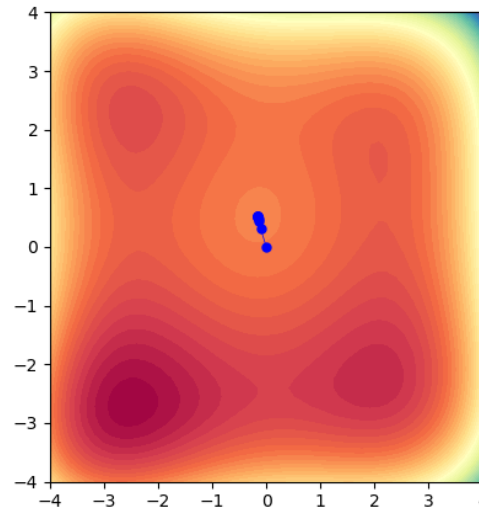
# Optimize.py for Simplex method (not working without tklib)

$$F(x,y) = -3.0 - 10x - 30x^2 + 1.5x^3 + 3x^4 + 30y - 30y^2 + 3y^4 + 3xy^2$$

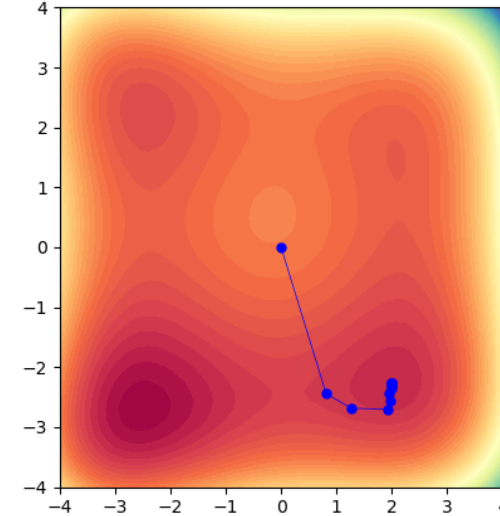
e.g.: python optimize.py 0.0 1.0 simplex



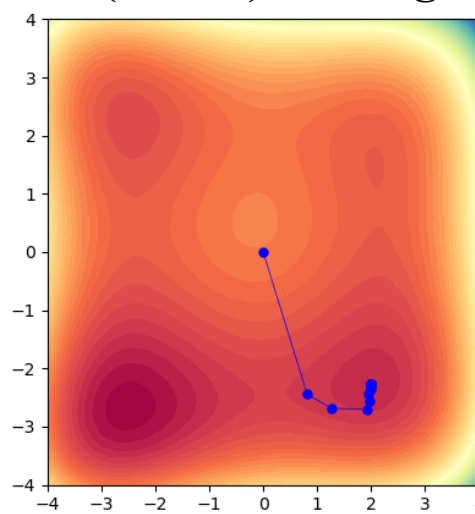
From (0.0 0.0) Newton



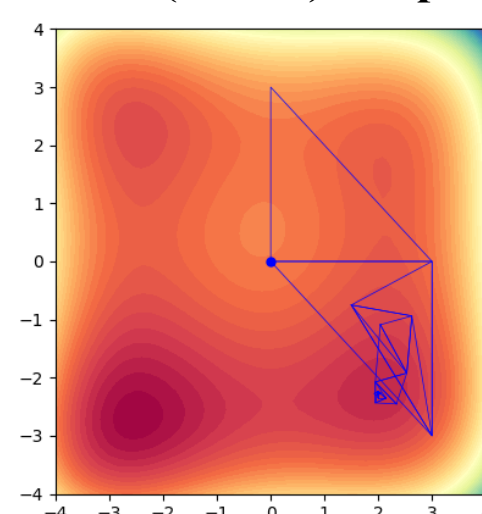
From (-1.0 -1.0) DFP golden



From (0.0 0.0) BFGS golden



From (0.0 1.0) Simplex



## Main algorism:

newton, sd, cg, broyden, dfp, bfgs  
simplex

## Direct search:

exact, one, simple  
armijo, golden

# Fourier transformation

## 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・ルベグの補題):  $\lim_{n \rightarrow \infty} c_n = 0$

# Fourier transformation

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 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

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

$f1, p1, A1 = 1.5, \pi/4.0, 1.0$

$f2, p2, A2 = 3.0, \pi/3.0, 0.3$

$f3, p3, A3 = 10.0, \pi/6.0, 0.5$

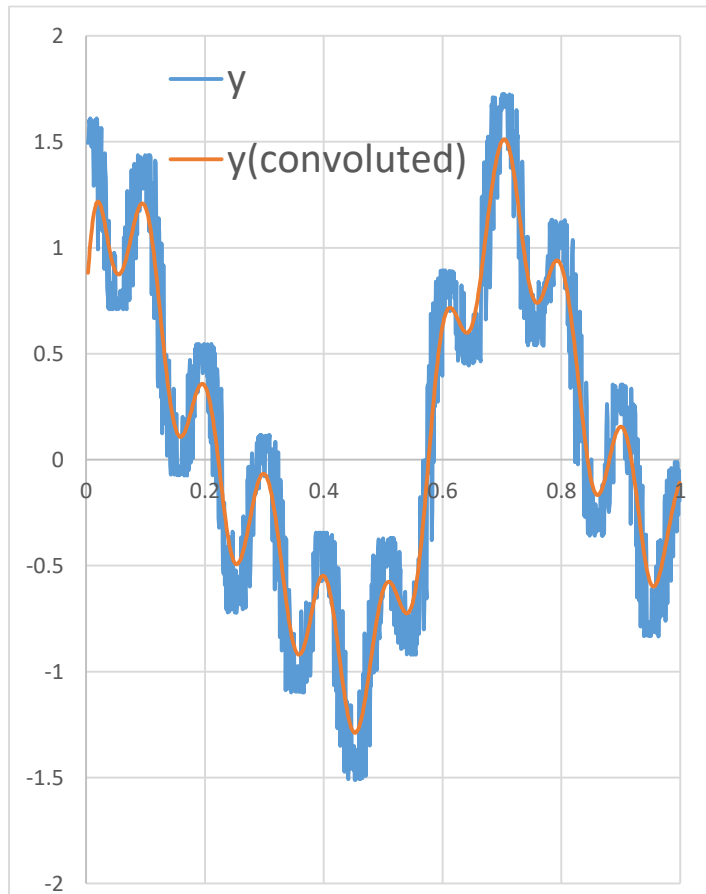
$x += \text{random}(0.03)$  # noise is simulated by random()

$y = A1 * \sin(2.0 * \pi * f1 * x + p1)$

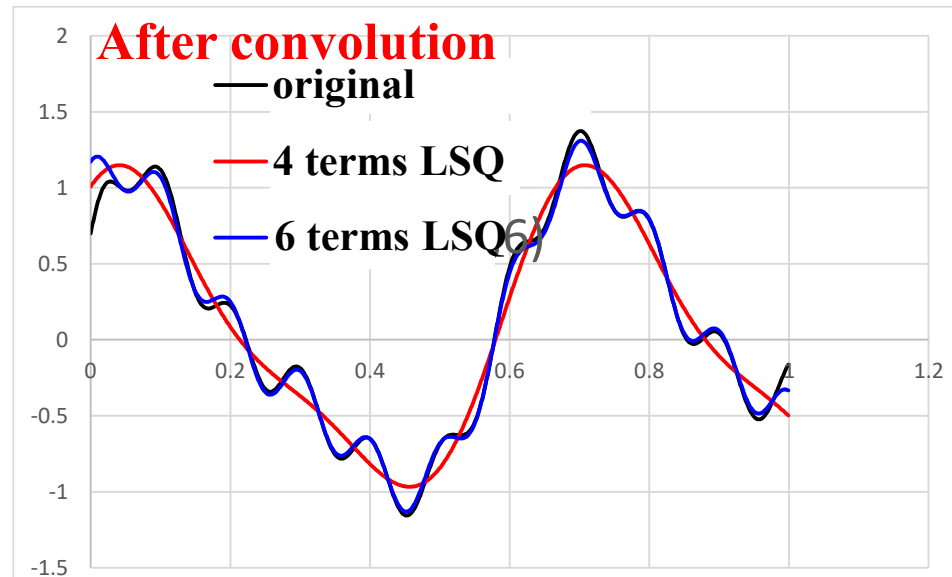
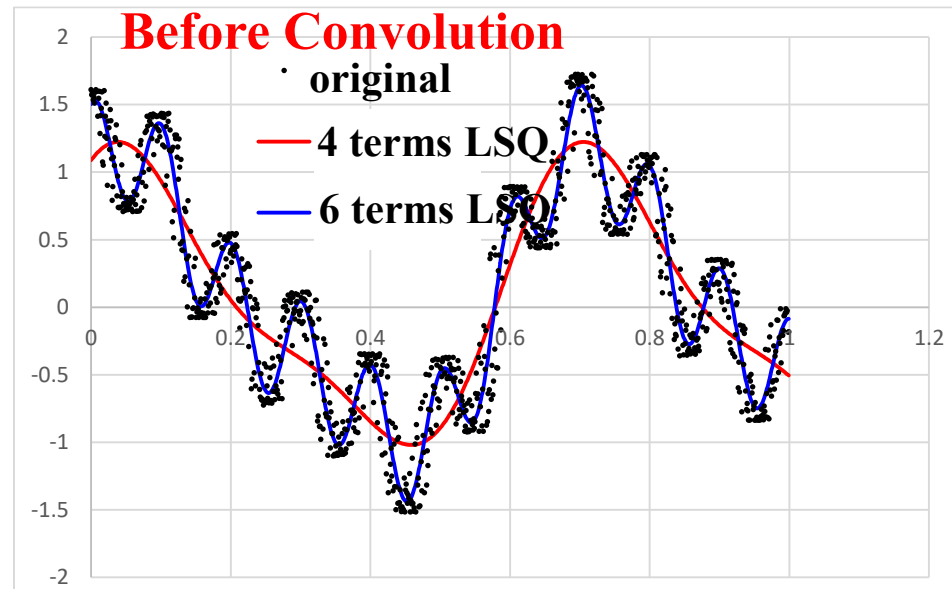
$+ A2 * \sin(2.0 * \pi * f2 * x + p2)$

$+ A3 * \sin(2.0 * \pi * f3 * x + p3)$

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

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

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)

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

$$\mathbf{y}_k = \sum_{j=0}^{N-1} \mathbf{x}_j \mathbf{w}_N^{kj}$$

$$\mathbf{w}_N = \exp(-i2\pi/N)$$

## DFT

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

## Inverse DFT

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

Using  $\mathbf{w}_N^k = \mathbf{w}_N^{k \bmod N}$  and  $\mathbf{w}_N^{k+N/2} = -\mathbf{w}_N^k$ , 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. FFT is identical calculation to DFT,  
but the calculation cost is proportional only to  $M \log N$  (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  $w_N^k = z$**

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

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

**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

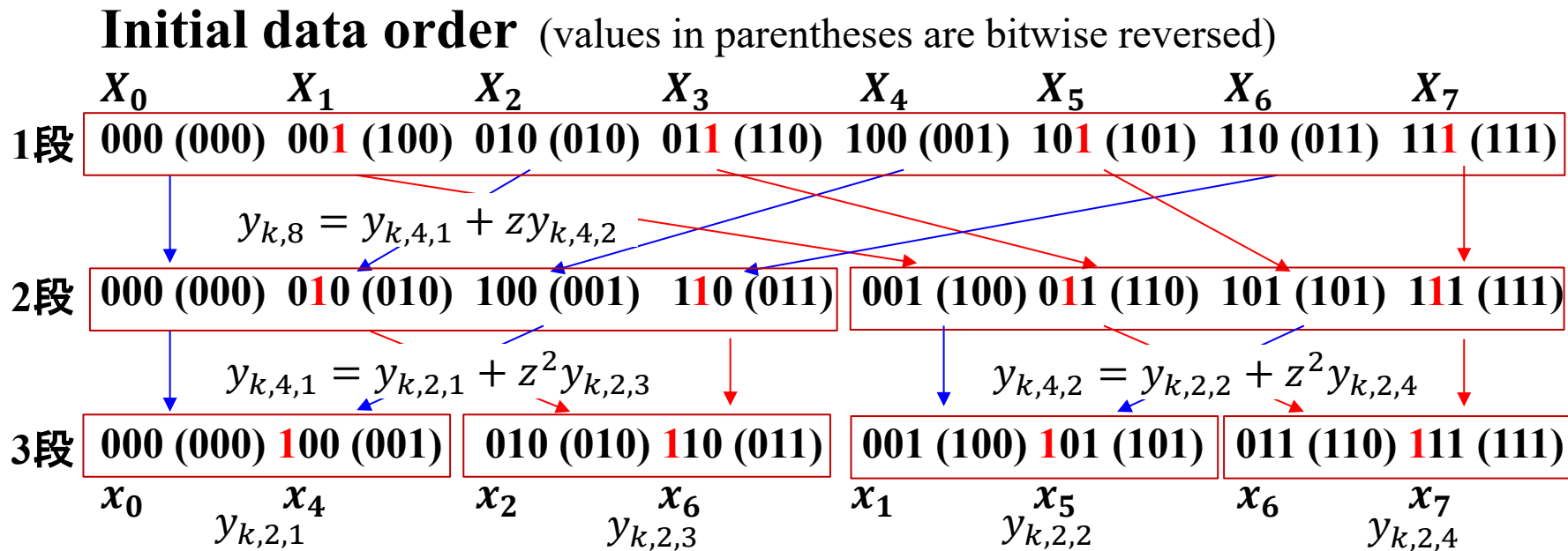
$N$  data series  $x_0 x_1 x_2 \cdots x_{N-1} \Rightarrow \text{FT: } X_0 X_1 X_2 \cdots X_{N-1}$

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



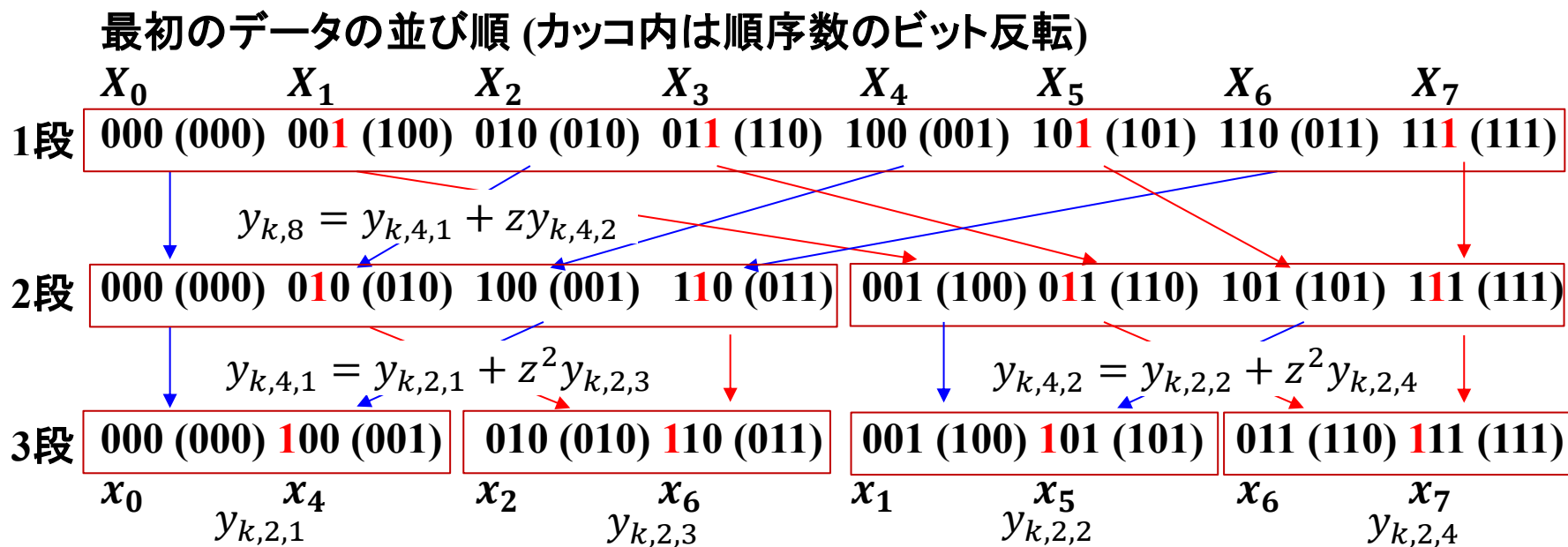
The order to sum up for FFT is different from the order of  $x_i$ .

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  $x_0 x_1 x_2 \cdots x_{N-1} \Rightarrow \text{FT: } X_0 X_1 X_2 \cdots X_{N-1}$   
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  
 $\Rightarrow$  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

# 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) ( $\sim x$ , 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.に戻って繰り返し

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.に戻って繰り返し

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.に戻って繰り返し

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.に戻って繰り返し

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:  $ret = 10011_2$

# 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** **1001100011110101101111011 1000000**

measure time to reverse **1001100011110101101111011<sub>2</sub>** 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
```

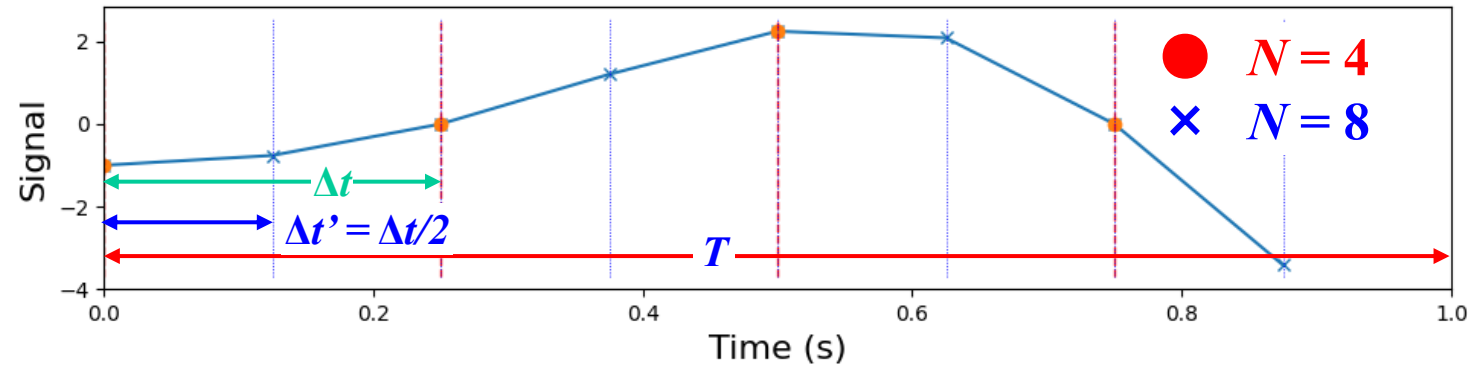
```
    return ret
```

fft.py

# FFT: python numpy.fft.fft()

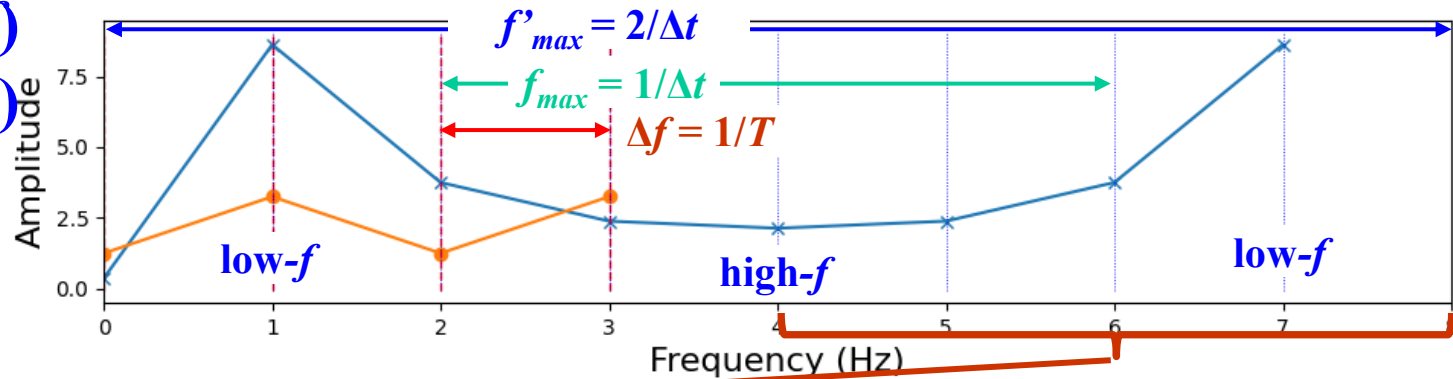
$f(t) = -\cos(2\pi t)(1+5t^2)$ , periodic in  $t = [0, 1)$

$f(t)$   
 $\text{tstep} = \text{tmax} / N$



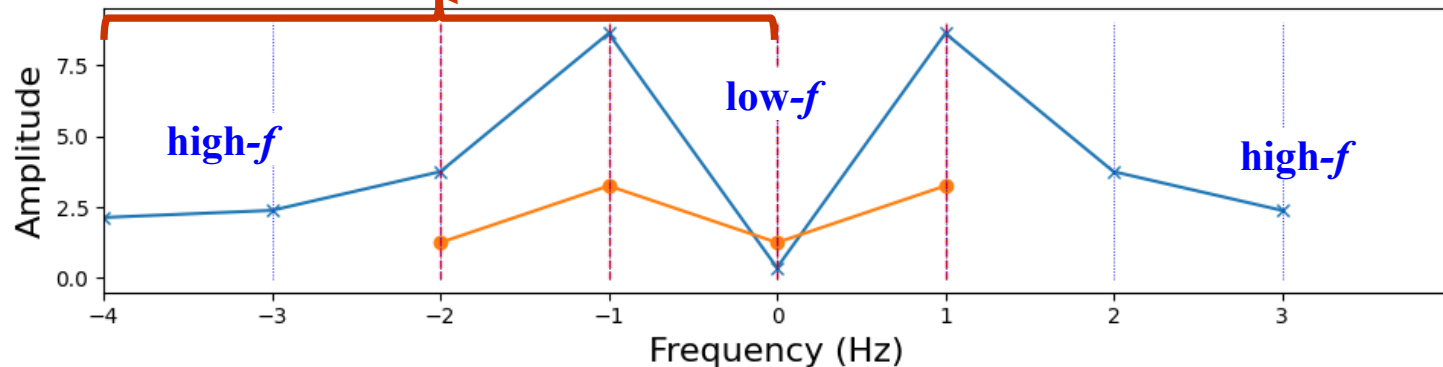
$F(f) = \text{np.fft.fft}(f)$

$f = \text{np.fft.fftfreq}(N, d=\text{tstep})$



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

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



NOTE:

Periodicity of Fourier func:

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

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

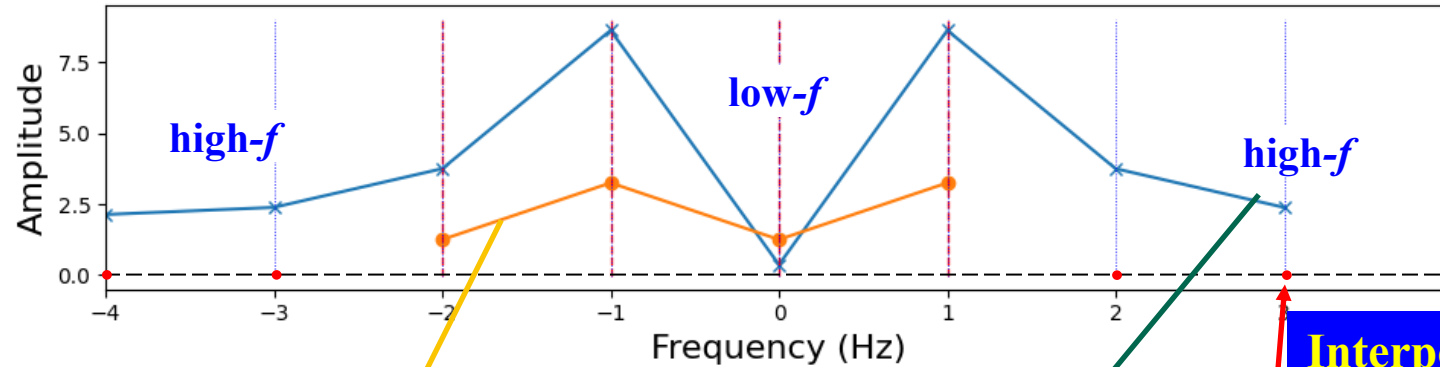
( $N = \text{len}(a)$ )

# 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

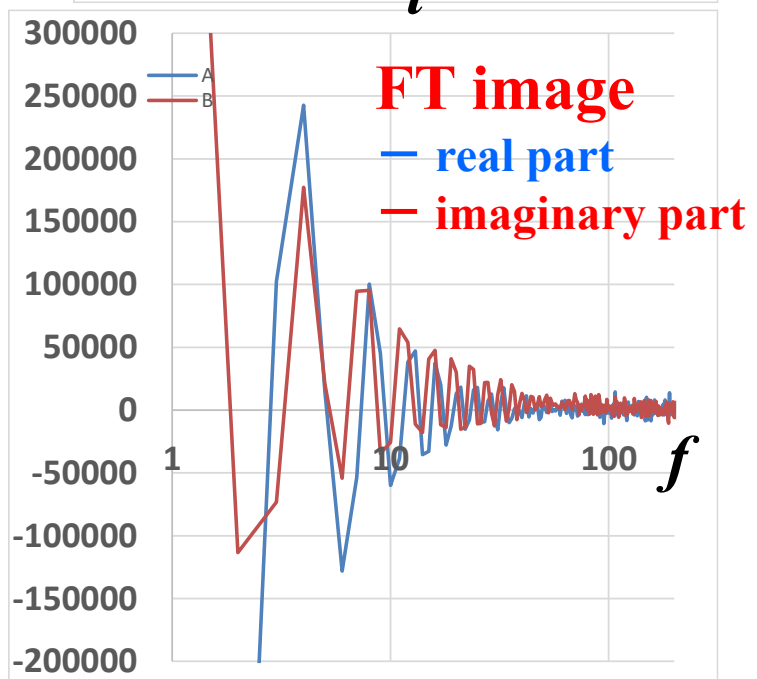
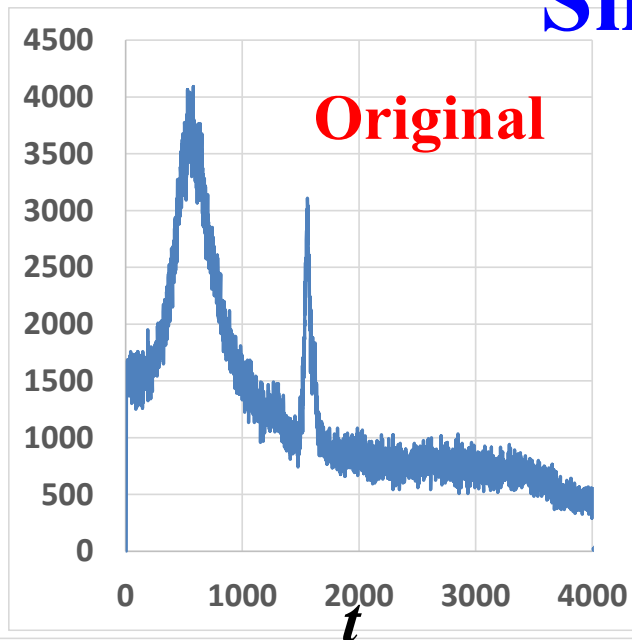
Interpolate by the FFT result for  $N = 4$ : e.g., to get  $f(t_{1/2} = \Delta t/2)$ :

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

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



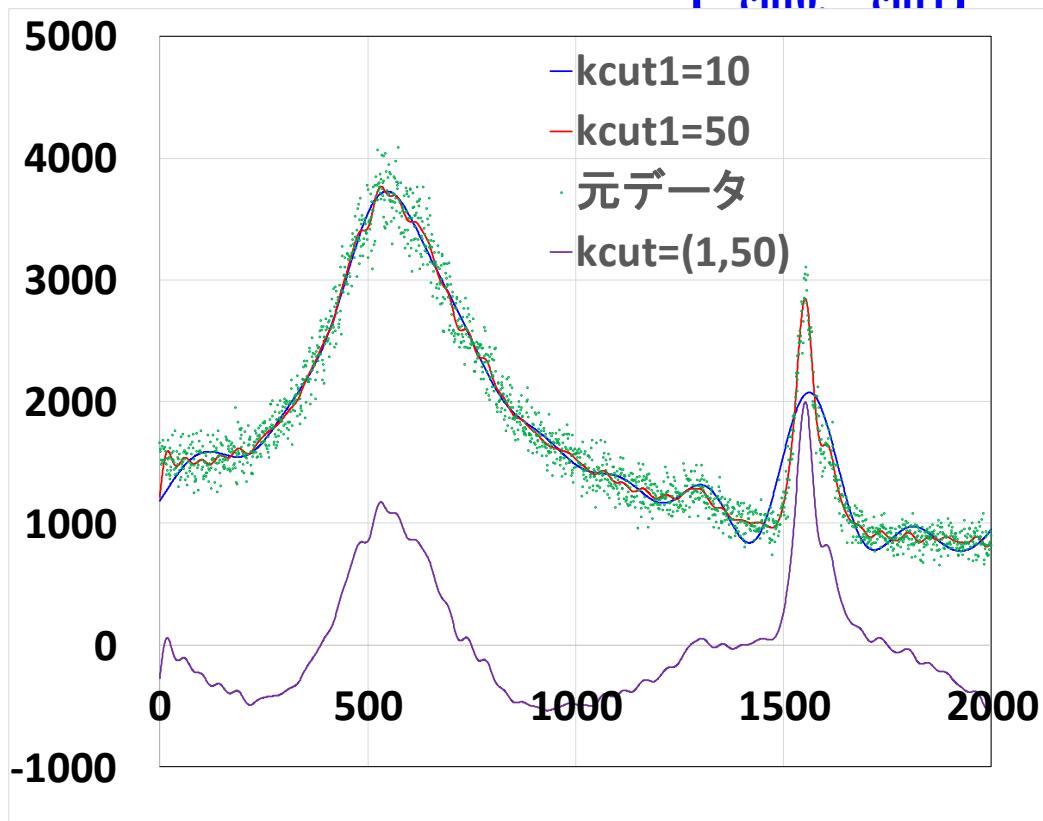
# Smoothing: FT



Remove high-frequency FT data: Smoothing  
Low-pass filter

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

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

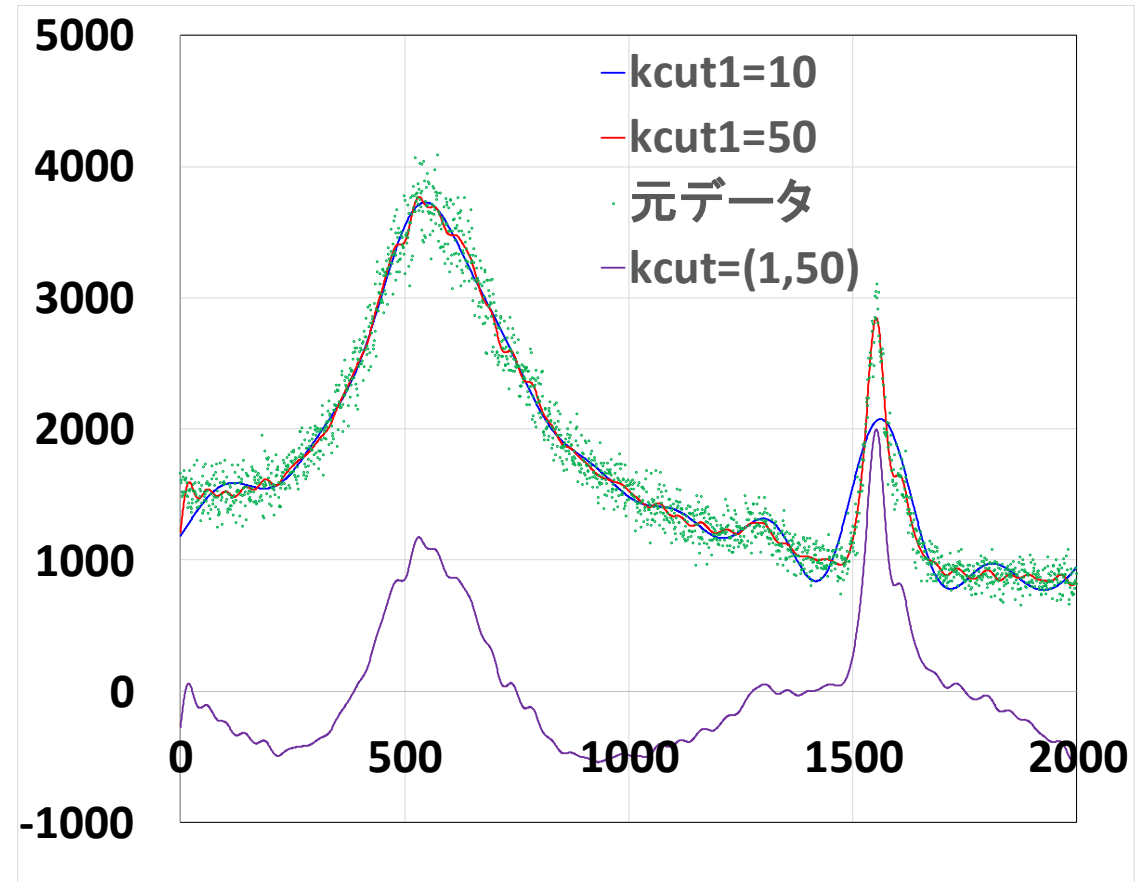


# Note:

**Be careful:** FFT high-pass filter can remove a baseline, but that baseline includes some signals

**Usual ways:**

1. Baseline function is optimized simultaneously with peaks.
3. Baseline function is determined from selected data where peaks do not affect.

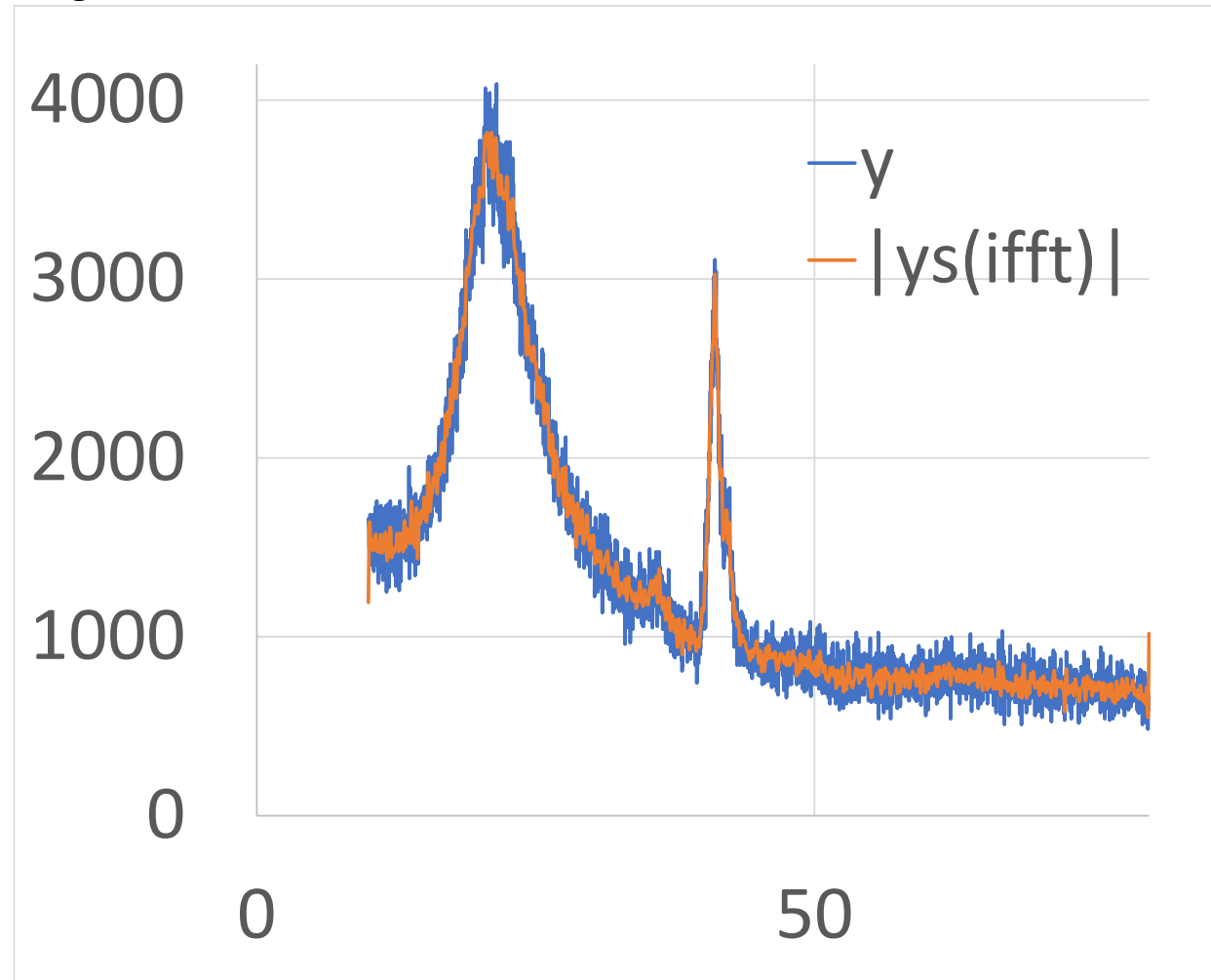


# Program: smoothing-fft.py

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

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

=> `plot smoothing-fft.csv`

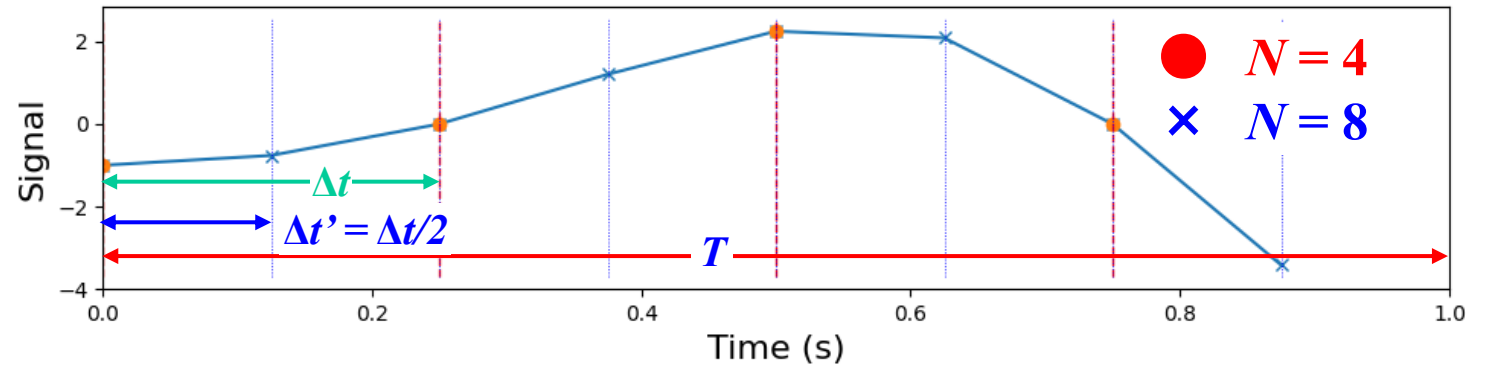


fft.py

# FFT: python numpy.fft.fft()

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

$f(t)$



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

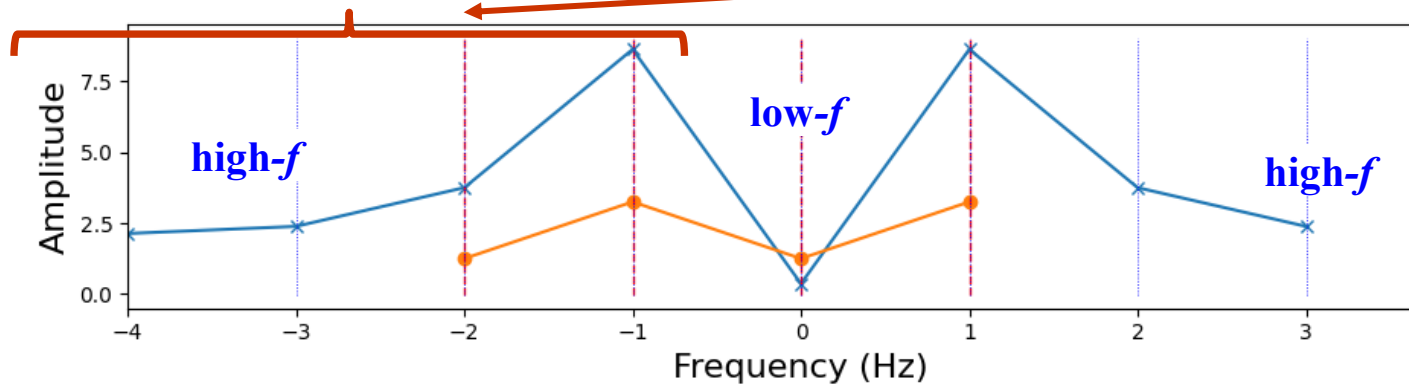
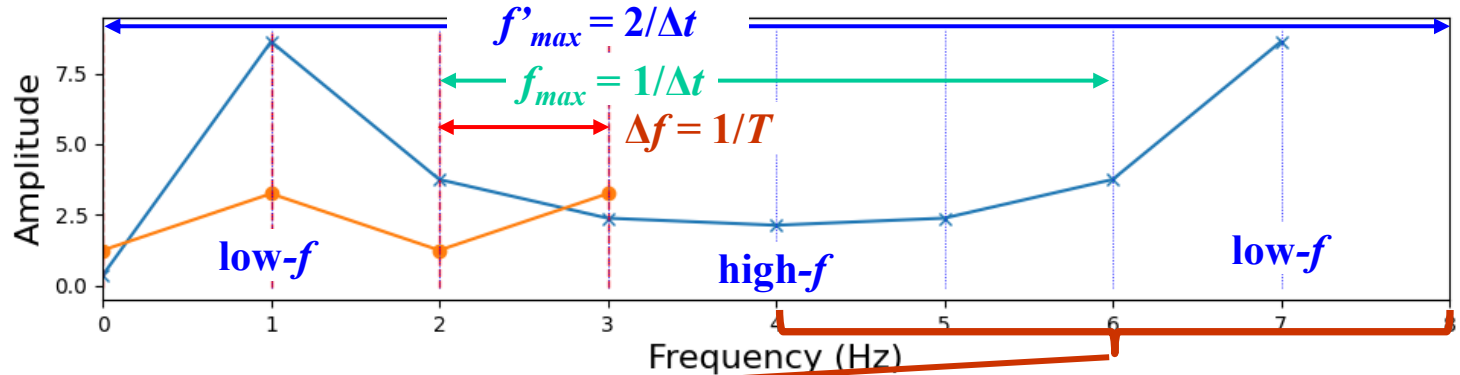
$F = \text{np.fft.fft}(f)$

$\text{Freq} = \text{np.fft.fftfreq}(N, d=t\_step)$

Centering:

$F2 = \text{np.fft.fftshift}(F)$

$\text{Freq2} = \text{np.fft.fftshift}(\text{freq})$



NOTE:

Periodicity of Fourier func:

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

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

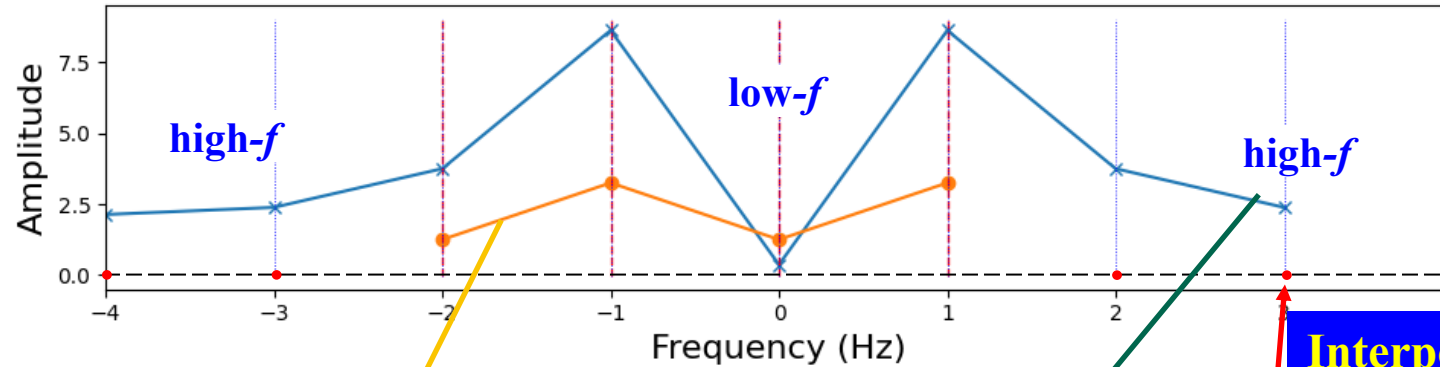
$(N = \text{len}(a))$

# 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 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 transformation

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

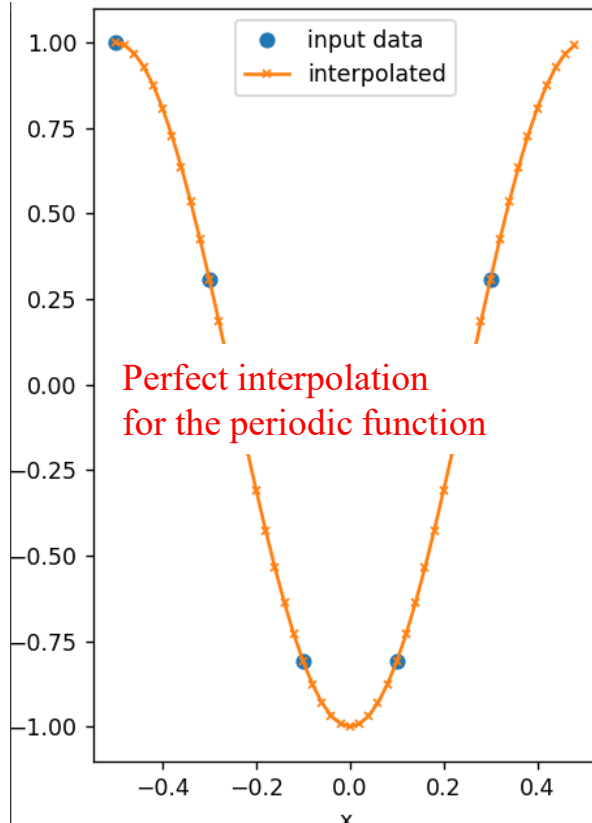
$$f(t_r = x\Delta t) = a_0 + a_{\pi/4} \mathbf{\exp\left(i \frac{\pi}{4} x\right)} + a_{2\pi/4} \mathbf{\exp\left(i \frac{2\pi}{4} x\right)} + a_{3\pi/4} \mathbf{\exp\left(i \frac{3\pi}{4} x\right)}$$

# Interpolation by FFT (Fast Fourier Transform)

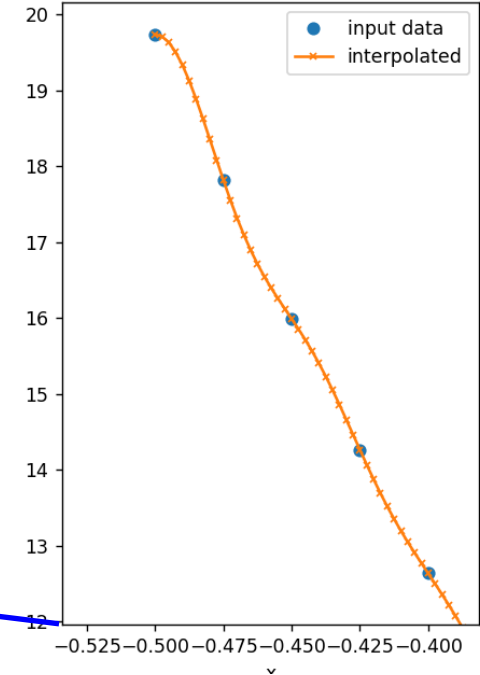
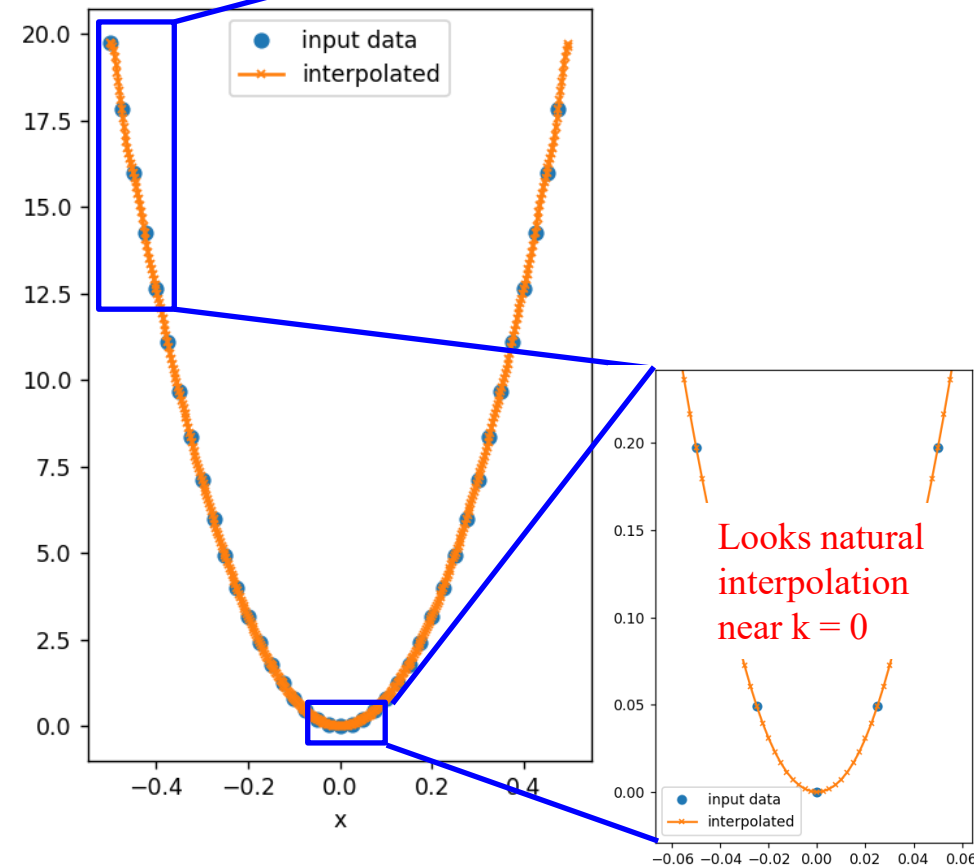
For a periodic function  $E(k)$  (like an energy function  $E(k)$  of crystal), interpolation can be carried out by:

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



# 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)

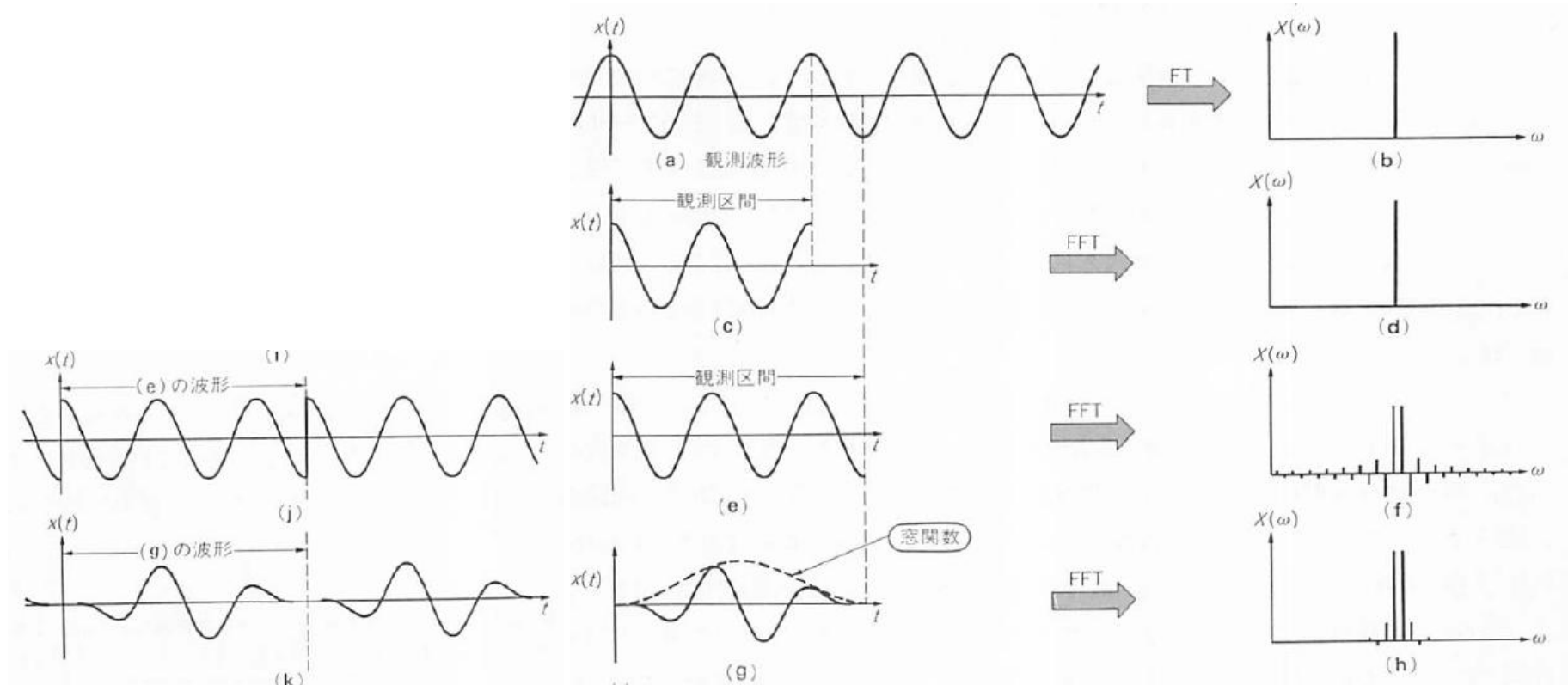
N	DFT1	DFT2	FFT	N log N
1024	1.32	2.41	1.87e-5	3080
2048	5.59	10.3	3.54e-5	6780
4096	23.6	47.7	6.62e-5	14800
8192	97.3	165	16.1e-5	32100



# 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 **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

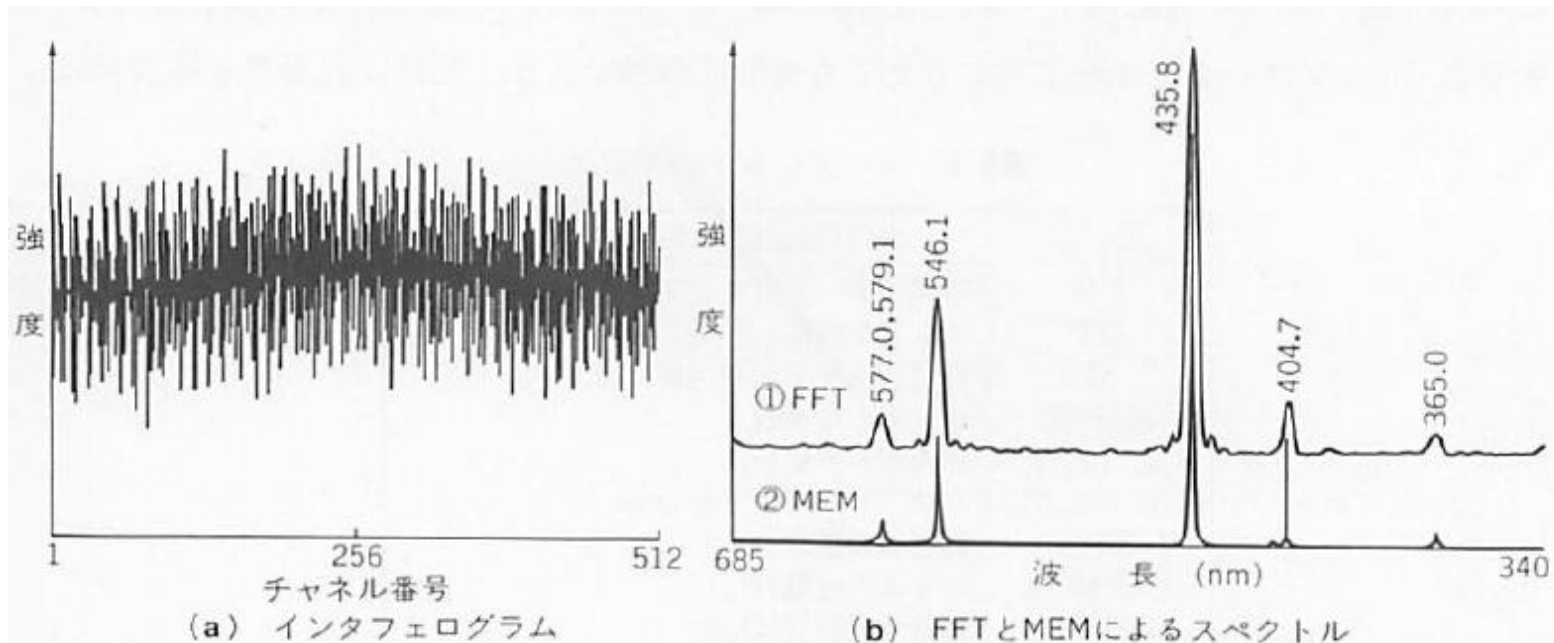
- Assume the lost data would have some constraints
- Use the concept of 'information entropy' and maximize it to estimate the spectrum
- Akaike's autoregressive model (赤池による自己回帰モデル) => identical to MEM

The order of the autoregressive model  $m$  must be determined

- So as to minimize Final Prediction Error (最終予測誤差)
- Algorithms: Burg method, etc

## Features

- Sharper spectrum than FFT
- Less ghost peaks and fringes



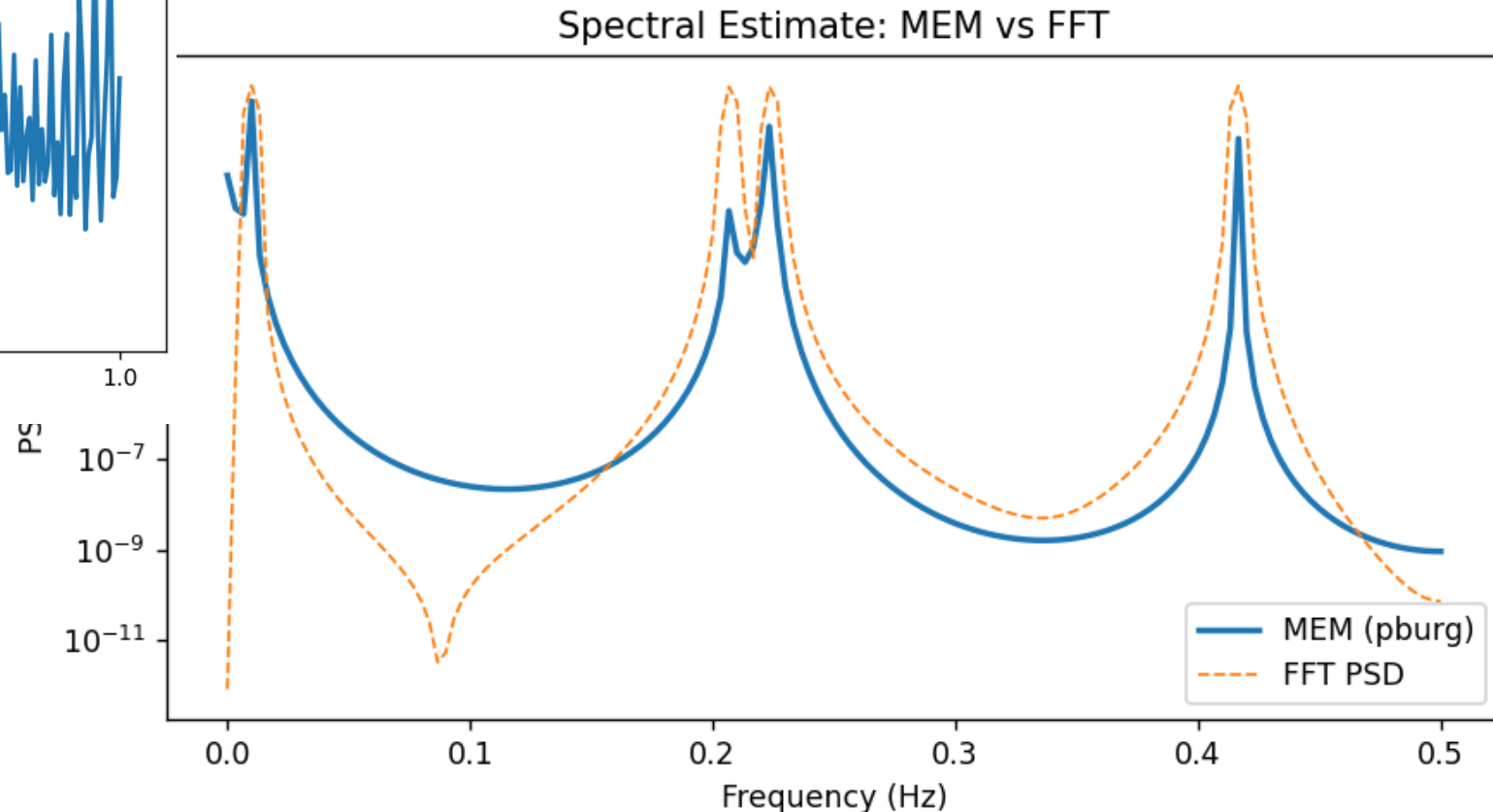
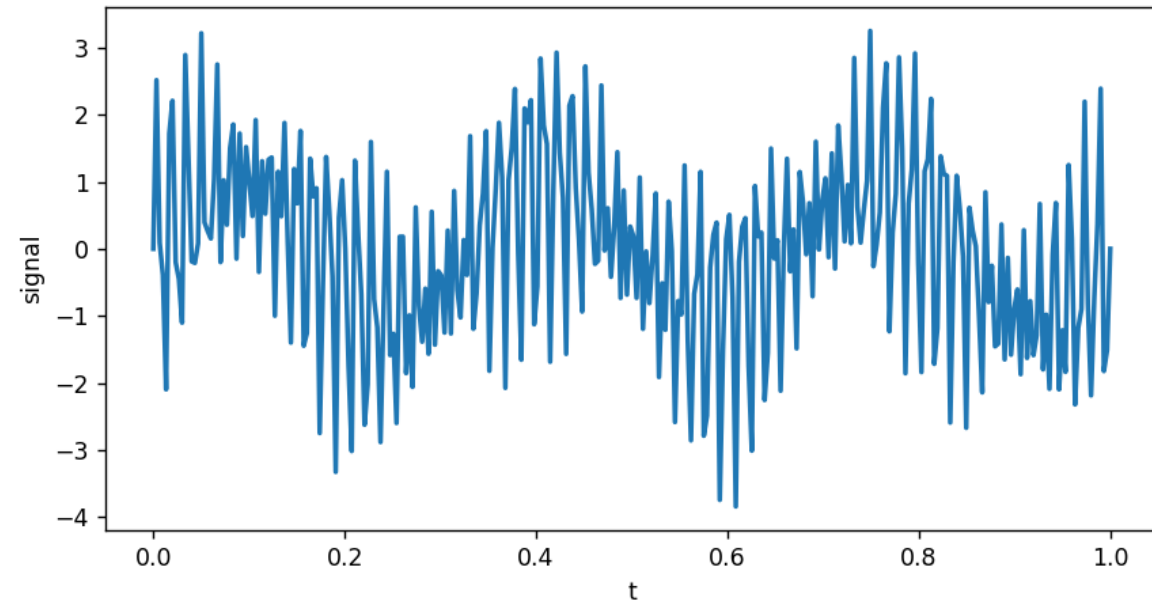
# Program: mem\_spectrum.py

> 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  $|\mathbf{G}_{hkl}| < G_{\max}$**

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

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

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

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

**=> Possibly speed up with GPU**

# Program: 1-D PW method

pw1d.py

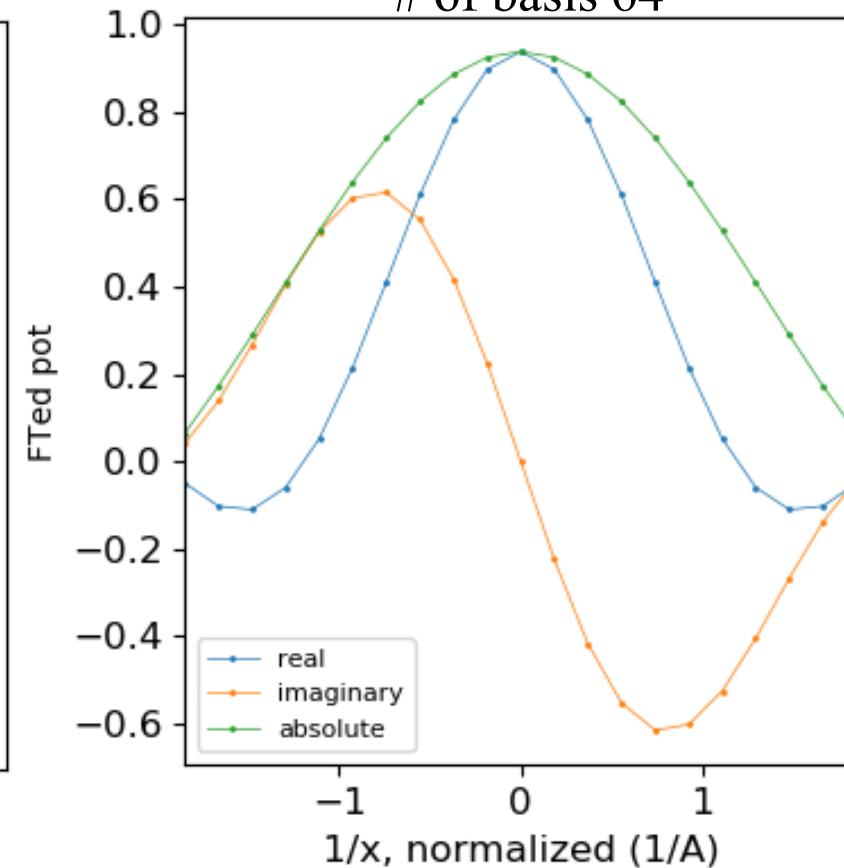
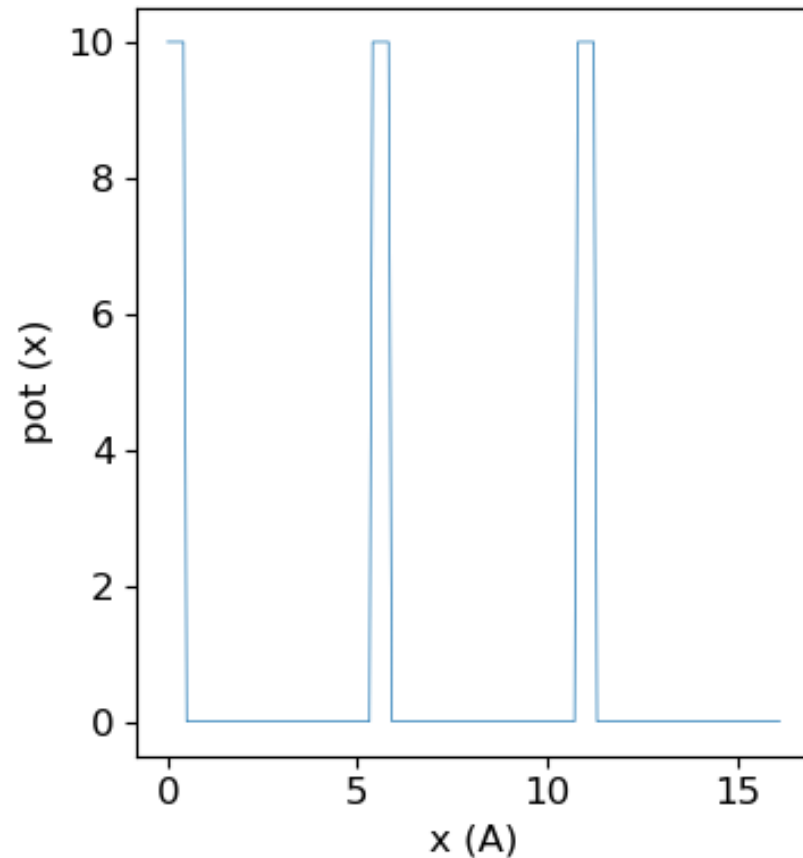
Lattice parameter (Si)  $a = 5.4064 \text{ \AA}$   $m^* = 1.0m_e$

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
```

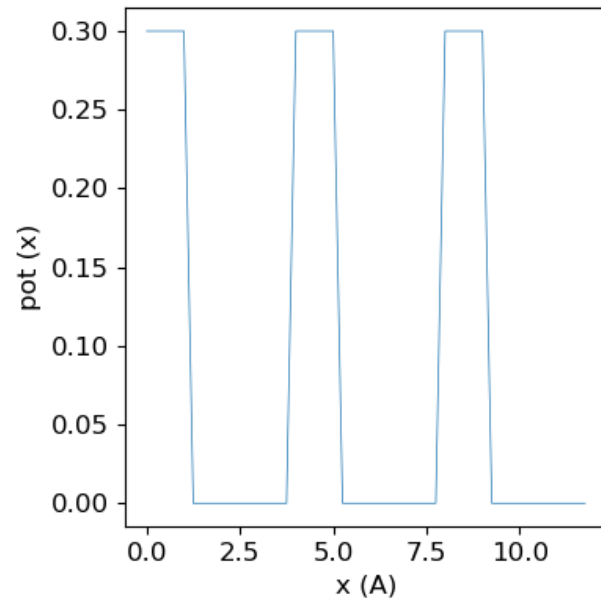
```
python pw1d.py band 5.4064 64 rect 0.5 10.0 3 -0.5 0.5 21
```

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

$a = 4.0 \text{ \AA}$

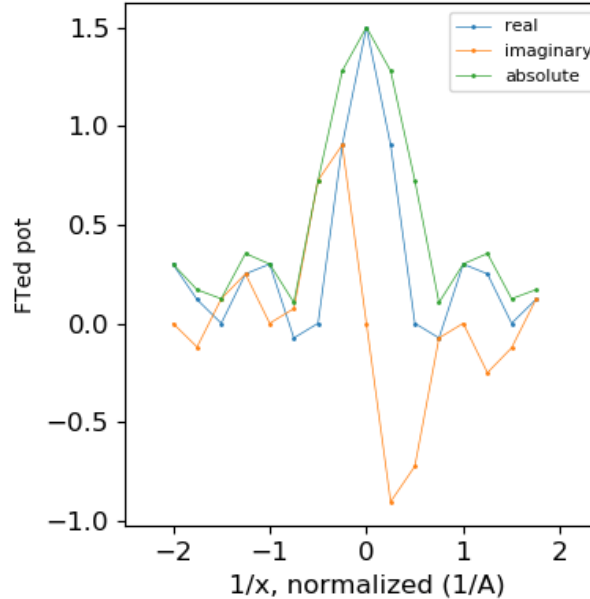
**potential  $V(x)$ :**

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



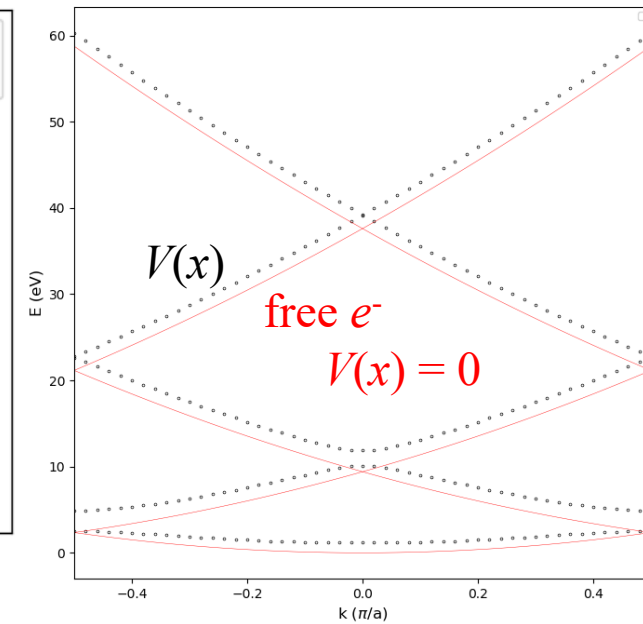
**FT coefficients of potential**

# of basis 16



**Band structure**

# of basis 5



**Matrix problems**

行列問題の解法



# Fundamental matrix operations

$C = A + B$ :

```
for ix in range(nx):  
    for iy in range(ny):  
         $c[ix][iy] = a[ix][iy] + b[ix][iy]$ ;
```

$C = A * B$ :

```
for ix in range(nx):  
    for iy in range(ny):  
         $c[ix][iy] = 0.0$ ;  
        for k in range(nk):  
             $c[ix][iy] = c[ix][iy] + a[ix][k] * b[k][iy]$ ;
```

To solve  $BC = A$

(i) Obtain  $B^{-1}$  and calculate  $B^{-1}A$

(ii) Directly solve  $BC = A$

=> Better to use open libraries

# Gauss elimination method (Gaussの消去法)

Upon a square matrix (正方行列)  $A$  and a vector  $B$  are given,  
solution of  $\mathbf{AX} = \mathbf{B}$  is obtained by  $X = A^{-1}B$ .

- Efficient for case more than one solutions for the same  $A$  and different  $B$ .
- Can produce roundoff errors and not efficient

=> Solve the linear simultaneous equations directly.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ \vdots & & & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

**Multiply  $a_{i1}/a_{11}$  ( $i = 2, 3, \dots, n$ ) to the first line and subtract it from  $i$ -th line  
=> make all  $a_{i1}$  ( $i \geq 2$ ) zero.**

Repeat this procedure for all the lines,  $A$  will be converted to upper-right triangle matrix (右上三角行列)

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}' & a_{23}' & \cdots & a_{2n}' \\ 0 & 0 & a_{33}' & & a_{3n}' \\ \vdots & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}' \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ \vdots \\ b_n' \end{pmatrix}$$

**Solve from the last line to upper lines, giving all  $x_i$**

**Note: Converting  $A$  to a band or triangle matrix enables solve the equation very easy**

# Row reduction method (掃き出し法)

Similar to the Gauss elimination method, but eliminates all non-diagonal terms

$$\begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22}' & 0 & \cdots & 0 \\ 0 & 0 & a_{33}' & & 0 \\ \vdots & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}' \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ \vdots \\ b_n' \end{pmatrix}$$

Obtain the solution by  $x_i = b_i' / a_{ii}'$

**Important: Regular matrix can be converted to triangle / band matrixes**

(正則行列は、適当な行列による変換で三角行列や帯行列に分解できる)

=> **ex. LU decomposition** (LU分解):  $A = LU$

**$L$ : Left-lower triangle,  $U$ : Right-upper triangle matrix**

## Solution of linear simul. eqs. : LU decomposition

- 1. Convert  $AX = B$  to  $LUX = B$  by  $A = LU$**
- 2. Solve  $LY = B$  to obtain  $Y$**
- 3. Solve  $UX = Y$  to obtain  $X$**

# Diagonalization of real symmetric matrix: Jacobi method (ヤコビ法)

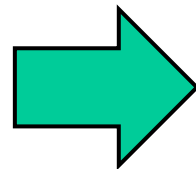
**Diagonalization of**  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$

=> can be done by conversion  $U^T A U$  with an orthogonal matrix (直交行列)  $U$

$$U = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$$\begin{aligned} U^T A U &= \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \\ &= \begin{pmatrix} a_{11} \cos^2 \theta + 2a_{12} \cos \theta \sin \theta + a_{22} \sin^2 \theta & (-a_{11} + a_{22}) \cos \theta \sin \theta + a_{12} (\cos^2 \theta - \sin^2 \theta) \\ (-a_{11} + a_{22}) \cos \theta \sin \theta + a_{12} (\cos^2 \theta - \sin^2 \theta) & a_{11} \sin^2 \theta - 2a_{12} \cos \theta \sin \theta + a_{22} \cos^2 \theta \end{pmatrix} \end{aligned}$$

$$(-a_{11} + a_{22}) \cos \theta \sin \theta + a_{12} (\cos^2 \theta - \sin^2 \theta) = 1/2 [(-a_{11} + a_{22}) \sin 2\theta + a_{12} \cos 2\theta] = 0$$



$$\theta = \pi / 4$$

$$a_{11} = a_{22}$$

$$\theta = (1/2) \tan^{-1} (2a_{12} / (a_{11} - a_{22})) \quad a_{11} \neq a_{22}$$

# Jacobi method

1. Choose the largest absolute value non-diagonal element  $a_{ij}$  in  $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{12} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{13} & a_{23} & a_{33} & & a_{3n} \\ \vdots & & & \ddots & \vdots \\ a_{1n} & a_{2n} & & \cdots & a_{nn} \end{pmatrix}$

2. Converting by  $A' = U^T A U$  with  $U = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & & & & & \vdots \\ \vdots & & \cos \theta & & -\sin \theta & & \vdots \\ \vdots & & & 1 & & & \vdots \\ \vdots & & \sin \theta & & \cos \theta & & \vdots \\ \vdots & & & & & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$  will give  $a_{ij}' = 0$

3. Choose the largest absolute value element  $a_{ij}'$  and repeat 2

=> The square sum of non-diagonal elements is reduce by a factor of  $2a_{ij}^2$

=> finite iterations will complete the diagonalization

**But it is hard to estimate the number of iterations required,  
and Jacobi method is not efficient for a large-size materix**

# Diagonalization of large-size matrix

## Householder method

1. Convert a symmetric matrix  $A$  to a triple diagonal matrix (三重対角行列)  $D$  using an orthogonal matrix (直交行列)  $U$

*Note: eigen values of  $U^T A U$  are equal to those of  $A$*

2. Solve eigen values of  $D$  by bisection method

## QR method

1. Regular  $n \times n$  matrix  $A$  is decomposed to  $A = QR$  (QR分解) using a regular orthogonal matrix  $Q$  and a right-upper matrix with positive diagonal elements  $R$ .
2.  $QR$ -decompose  $A_k$ :  $A_k = Q_k R_k$
3. Convert  $A_k$  to  $A_{k+1} = Q_k^T A_k Q_k = R_k Q_k$  (similar transformation, 相似変換)
4. Repeating 2 and 3 will converge  $A_k$  to a right-upper triangle matrix  $A_R$   
=> Solve eigen values of  $A_R$

*If  $A$  is a symmetric matrix,  $A_R$  will be a diagonal matrix.*

# Linear algebra libraries

## (線形幾何学・行列計算ライブラリ)

Fortran, C, C++, etc

LAPACK (Linear Algebra PACKage)

ScaLAPACK (Scalable LAPACK)

Intel Math Kernel Library (MKL)

One API: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html>

Python: numpy.linalg, scipy.linalg

### matrix.py

<b>Product of matrixes</b>	<b>AB</b>	: C	= A @ B
<b>Inner product</b>	<b>V1・V2</b>	: inner	= numpy.dot(V1, V2)
		inner	= numpy.inner(V1, V2)
<b>Outer product</b>	<b>V1 × V2</b>	: V3	= numpy.cross(V1, V2)
<b>Inverse matrix</b>		: Ai	= numpy.linalg.inv(A)
<b>Determinant</b>		: det	= numpy.linalg.det(A)
<b>Eigen values/vectors</b>		: lA, vA	= numpy.linalg.eig(A)
<b>Solve simul. linear eqs.</b>	<b>AX = B</b>	: X	= numpy.linalg.solve(A, B)
<b>LU decomposition</b>		: P, L, U	= numpy.linalg.lu(A)
<b>Cholesky decomposition</b>	<b>A=LL<sup>T</sup></b>	: L	= numpy.linalg.cholesky(A)
<b>QR decomposition</b>	<b>A=QR</b>	: Q, R	= scipy.linalg.qr(A)

# numpy.ndarrayでの行列計算

- numpy.ndarray()型では直接の行列演算が可能。  
しかし、演算子、関数に注意

- 同じ次元の行列・ベクトルの和・差

**+, - 演算子**を使う

- 行列、ベクトルとの積

**@ 演算子**を使う

## 1次元配列 (ベクトル) が入る場合は注意

$va1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$  (縦ベクトル)      $va1 = [1, 2]$  (横ベクトル)

- ベクトル同士の積では、**縦ベクトルと横ベクトル**[1 2]を区別しない:

$va1 @ va2$  で内積計算



# その他の行列計算

内積  $\text{val} @ \text{val}$

`np.inner(val, va2)`

`np.dot(val, va2)`

# これらをお勧め

外積

`np.cross(val, va2)`

Outer product `np.outer(val, va2)` # 外積ではなく、要素ごとの積

逆行列 `np.linalg.inv(ma)`

行列式 `np.linalg.det(ma)`

固有値 `np.linalg.eigvals(ma)`

# 解が複素数の場合は複素数型配列で返ってくる

一次連立方程式  $\text{ma} @ \text{X} = \text{val}$  の解 `X = solve(ma, val)`

# 行列計算の応用: ベンゼンの電子準位

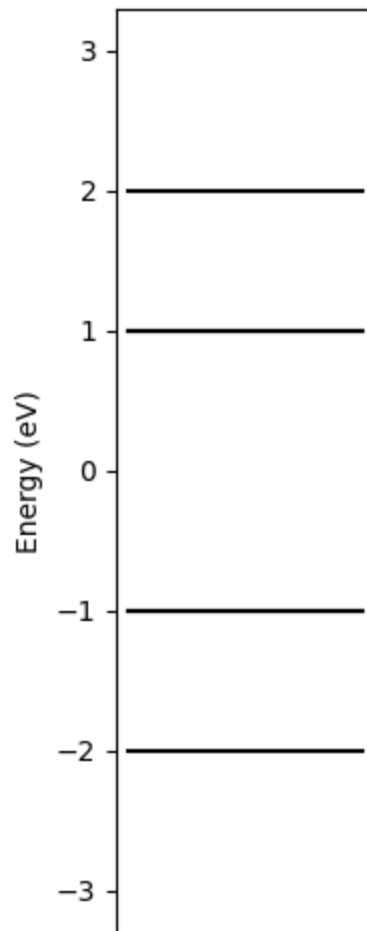
Hückel近似を使う: benzene.py

C 2pのエネルギー準位を  $\varepsilon_{2p} = 0$ 、  
共鳴積分を  $\beta_{2p\pi-\pi} = 1$   
としたときの

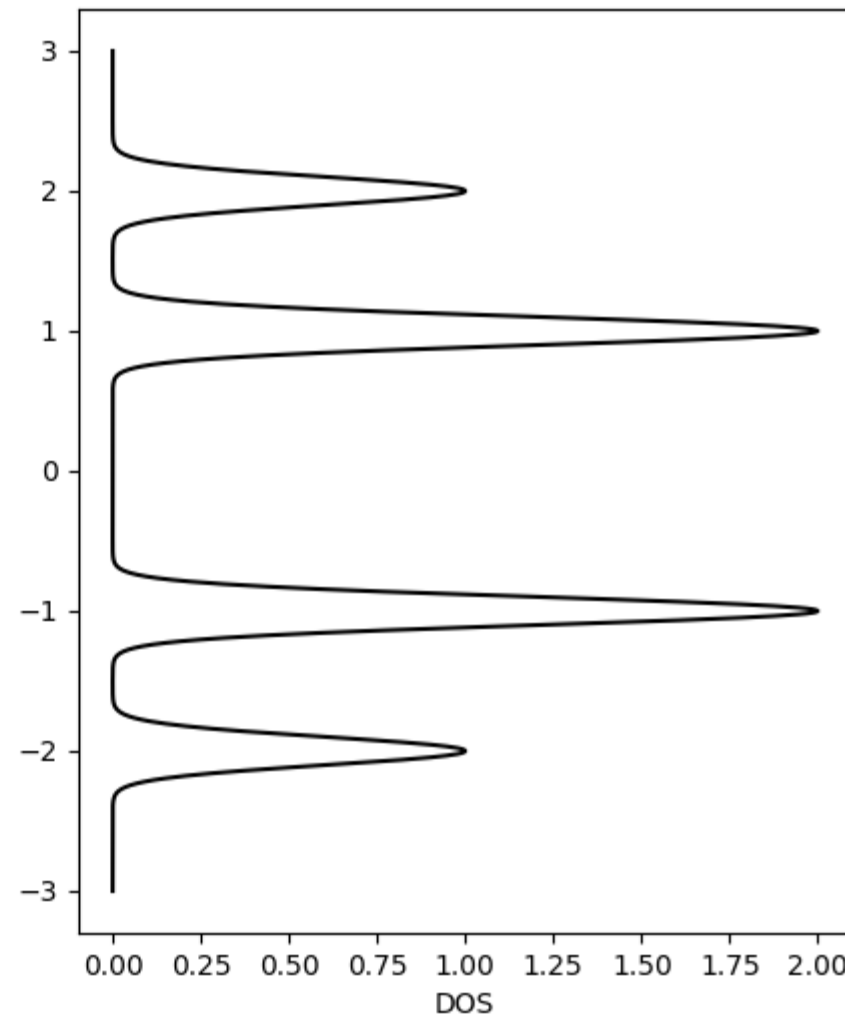
ハミルトニアンを対角化して  
エネルギー準位を計算できる

```
H = np.array([[0, 1, 0, 0, 0, 1],  
              [1, 0, 1, 0, 0, 0],  
              [0, 1, 0, 1, 0, 0],  
              [0, 0, 1, 0, 1, 0],  
              [0, 0, 0, 1, 0, 1],  
              [1, 0, 0, 0, 1, 0]])
```

エネルギー準位図



状態密度 (Density of states)

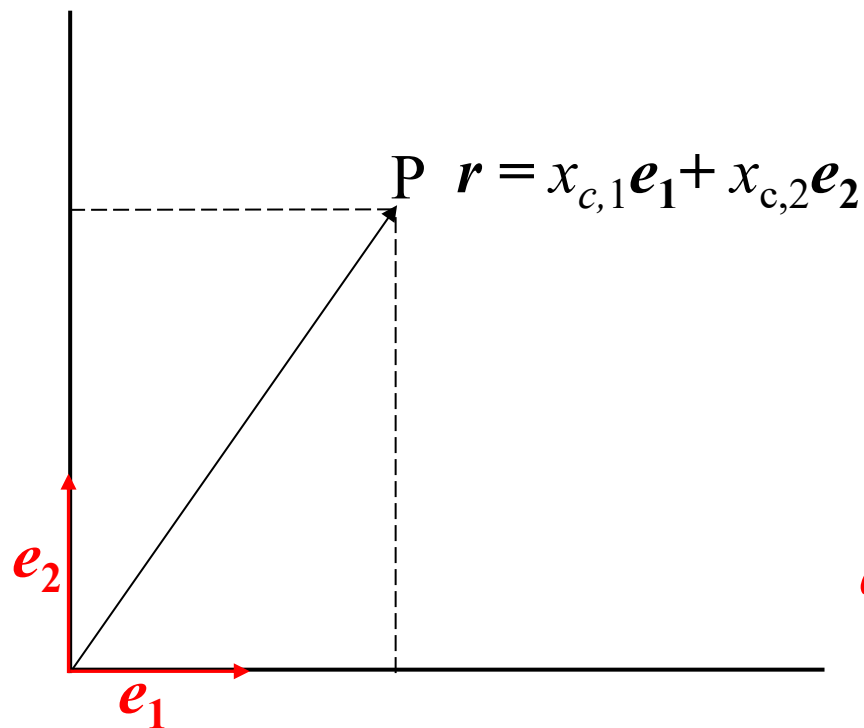


# Applications

## 応用

# 一般座標系 (general coordinate system)

直交座標系 (Orthogonal)  
デカルト座標系 (Cartesian)

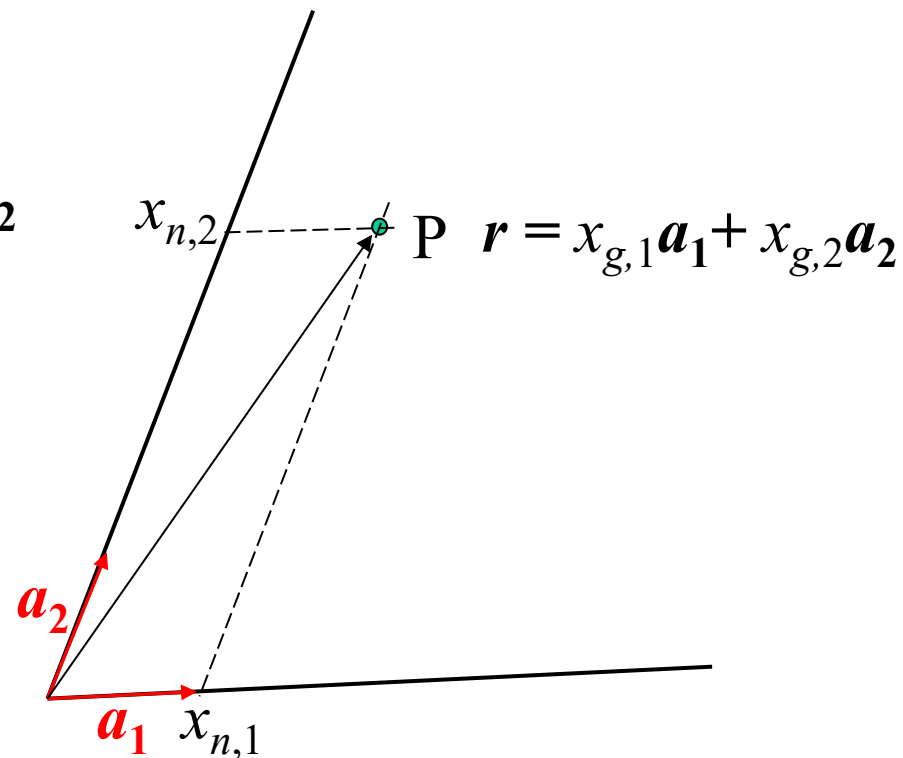


正規直交系 (orthonormal system)

$$\mathbf{e}_i \cdot \mathbf{e}_j = \delta_{ij}$$

$$|\mathbf{e}_i| = 1$$

一般座標/非直交系 (Non-Cartesian)



一般座標系 (general coordinate system)

$$\mathbf{a}_i \cdot \mathbf{a}_j \neq \delta_{ij}$$

$\mathbf{e}_i, \mathbf{a}_i$ : 基底ベクトル (base vector)

# Cartesian – general coord. Conversion

(直交系 – 一般座標系變換)

$$\mathbf{r} = x_{c,1}\mathbf{e}_1 + x_{c,2}\mathbf{e}_2 = x_{g,1}\mathbf{a}_1 + x_{g,2}\mathbf{a}_2$$

$$x_{c,1} = x_{g,1} \mathbf{a}_1 \cdot \mathbf{e}_1 + x_{g,2} \mathbf{a}_2 \cdot \mathbf{e}_1$$

$$x_{c,2} = x_{g,1} \mathbf{a}_1 \cdot \mathbf{e}_2 + x_{g,2} \mathbf{a}_2 \cdot \mathbf{e}_2$$

If  $\mathbf{a}_1 = a_{11}\mathbf{e}_1 + a_{12}\mathbf{e}_2$

$\mathbf{a}_2 = a_{21}\mathbf{e}_1 + a_{22}\mathbf{e}_2$

are given,

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$$

$$\begin{matrix} x_{c,1} = x_{g,1}a_{11} + x_{g,2}a_{21} \\ x_{c,2} = x_{g,1}a_{12} + x_{g,2}a_{22} \end{matrix} \quad \begin{pmatrix} x_{c,1} \\ x_{c,2} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} x_{g,1} \\ x_{g,2} \end{pmatrix}$$

# Fractional coordinates in crystal

(結晶の内部座標)

Lattice parameters:  $a, b, c$  ( $= a_1, a_2, a_3$ ),  $\alpha, \beta, \gamma$  ( $= \alpha_{23}, \alpha_{13}, \alpha_{12}$ )

Lattice vectors:  $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3 = \mathbf{a}, \mathbf{b}, \mathbf{c}$

$$\mathbf{r} = x_{f,1}\mathbf{a}_1 + x_{f,2}\mathbf{a}_2 + x_{f,3}\mathbf{a}_3 = x_{c,1}\mathbf{e}_1 + x_{c,2}\mathbf{e}_2 + x_{c,3}\mathbf{e}_3$$

$(x_{f,1}, x_{f,2}, x_{f,3})$ : Fractional coordinate (部分座標)

Internal coordinate (内部座標)

$$|\mathbf{a}_i| = a_i$$

$$\mathbf{a}_i \cdot \mathbf{a}_j = a_i a_j \cos \alpha_{ij} \quad (i \neq j)$$

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{pmatrix}$$

Fractional coordinate to Cartesian coordinate

$$\begin{pmatrix} x_{c,1} \\ x_{c,2} \\ x_{c,3} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \begin{pmatrix} x_{f,1} \\ x_{f,2} \\ x_{f,3} \end{pmatrix}$$

# Conversion matrix

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{pmatrix}$$

$$|\mathbf{a}_i| = a_i$$

$$\mathbf{a}_i \cdot \mathbf{a}_j = \cos \alpha_{ij} \quad (i \neq j)$$

$$a, b, c \quad (= a_1, a_2, a_3)$$

$$\alpha, \beta, \gamma \quad (= \alpha_{23}, \alpha_{13}, \alpha_{12})$$

**tkcrystalbase.cal\_lattice\_vectors()**

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ b \cos \gamma & b \sin \gamma & 0 \\ c \cos \beta & c \frac{\cos \alpha - \cos \beta \cos \gamma}{\sin \gamma} & a_{33} \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{pmatrix}$$

$$a_{33} = \sqrt{c^2 - a_{31}^2 - a_{32}^2}$$

# Lattice properties

## Unit cell volume

$$V = \mathbf{a}_1 \cdot (\mathbf{a}_2 \times \mathbf{a}_3) \quad \text{tkcrystalbase.cal\_volume ()}$$

## Distance $\mathbf{r}_{kl} = \mathbf{r}_k - \mathbf{r}_l$ tkcrystalbase.distance2() / .distance()

$$r_{kl}^2 = |\mathbf{r}_{kl}|^2 = \sum_{i=0}^2 \sum_{j=0}^2 \mathbf{a}_i \cdot \mathbf{a}_j x_{kl,i} x_{kl,j} = \sum_{i,j} g_{ij} x_{kl,i} x_{kl,j}$$

$$g_{ij} = \mathbf{a}_i \cdot \mathbf{a}_j: \text{Metric tensor (計量テンソル)}$$

$$\text{tkcrystalbase.cal\_metrics()}$$

## Reciprocal lattice vectors tkcrystalbase.cal\\_reciprocal\\_lattice\\_vectors()

$$\mathbf{a}_1^* = \mathbf{a}_2 \times \mathbf{a}_3 / V$$

$$\mathbf{a}_2^* = \mathbf{a}_3 \times \mathbf{a}_1 / V$$

$$\mathbf{a}_3^* = \mathbf{a}_1 \times \mathbf{a}_2 / V$$

## Reciprocal vector at $(h \ k \ l)$

$$\mathbf{G}_{hkl} = h\mathbf{a}_1^* + k\mathbf{a}_2^* + l\mathbf{a}_3^*$$

## Lattice space

$$d_{hkl}^{-2} = |\mathbf{G}_{hkl}|^2 = \sum_{i=0}^3 \sum_{j=0}^3 \mathbf{a}_i^* \cdot \mathbf{a}_j^* h_i h_j = \sum_{i,j} Rg_{ij} h_i h_j$$

## Bragg angle

$$2d_{hkl} \sin \theta = \lambda$$

$$h, k, l \quad (= h_1, h_2, h_3)$$

$$Rg_{ij} = \mathbf{a}_i^* \cdot \mathbf{a}_j^*$$



# Inter-atomic distances

python crystal\_distance.py

NaCl

Lattice parameters: [5.62, 5.62, 5.62, 90.0, 90.0, 90.0]

Lattice vectors:

ax: ( 5.62, 0, 0) A

ay: ( 2.546e-10, 5.62, 0) A

az: ( 2.546e-10, 0, 5.62) A

Metric tensor:

gij: ( 31.58, 1.431e-09, 1.431e-09) A

( 1.431e-09, 31.58, 6.48e-20) A

( 1.431e-09, 6.48e-20, 31.58) A

Volume: 177.5 A<sup>3</sup>

Unit cell volume: 177.5 A<sup>3</sup>

Reciprocal lattice parameters: [0.17793594306049823, 0.17793594306049823, 0.17793594306049823, 90.00000000257246, 90.00000000516778, 90.00000000516778]

Reciprocal lattice vectors:

Rax: ( 0.1779, -8.06e-12, -8.06e-12) A<sup>-1</sup>

Ray: ( 0, 0.1779, 0) A<sup>-1</sup>

Raz: ( 0, 0, 0.1779) A<sup>-1</sup>

Reciprocal lattice metric tensor:

Rgij: ( 0.03166, -1.422e-12, -1.422e-12) A<sup>-1</sup>

(-1.422e-12, 0.03166, 6.382e-23) A<sup>-1</sup>

(-1.422e-12, 6.382e-23, 0.03166) A<sup>-1</sup>

Reciprocal unit cell volume: 0.005634 A<sup>-3</sup>

nmax: 1 1 1

Interatomic distances:

Cl1 ( 0.5, 0, 0) - Na4 ( 0.5, 0.5, 0) + ( 0, -1, 0): dis = 2.81 A

(cut)

Na4 ( 0.5, 0.5, 0) - Na1 ( 0, 0, 0) + ( 0, 1, 0): dis = 3.974 A

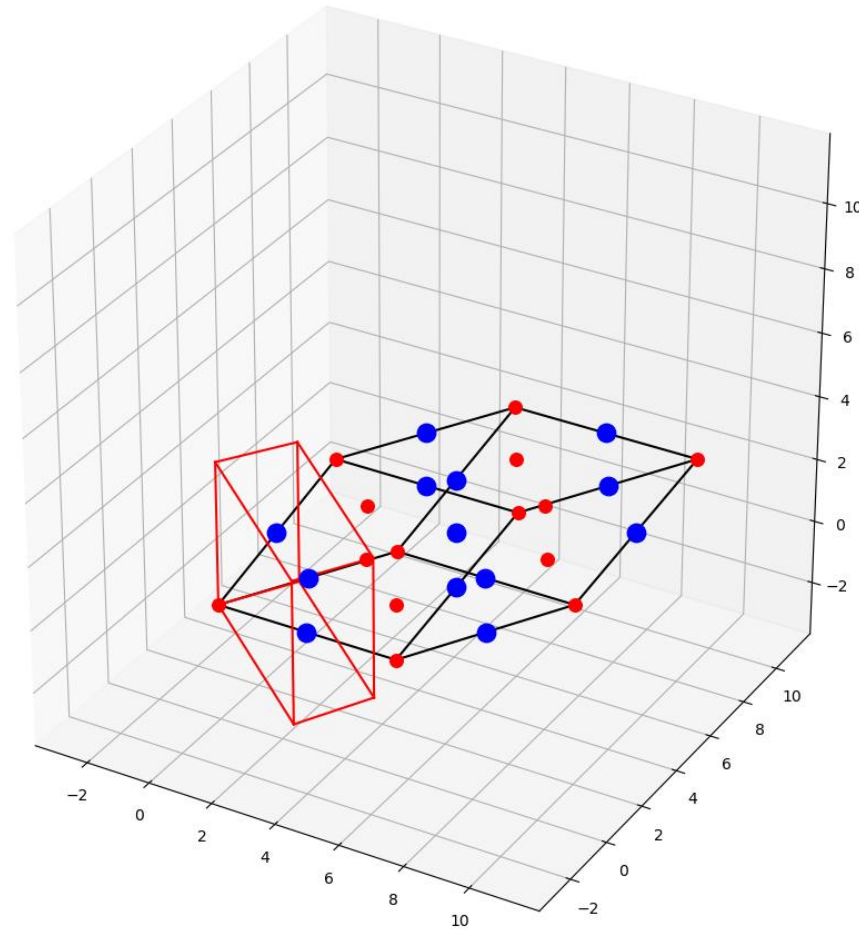
Na4 ( 0.5, 0.5, 0) - Na2 ( 0, 0.5, 0.5) + ( 1, 0, -1): dis = 3.974 A

Na4 ( 0.5, 0.5, 0) - Na1 ( 0, 0, 0) + ( 1, 0, 0): dis = 3.974 A

# Fractional – Cartesian conversion

`python crystal_draw_cell.py`

Rhombohedral cell  
and reciprocal unit cell



# Bragg angles

NaCl

python crystal\_xrd.py

Lattice parameters: [5.62, 5.62, 5.62, 90.0, 90.0, 90.0]

Lattice vectors:

ax: ( 5.62, 0, 0) Å

ay: ( 2.546e-10, 5.62, 0) Å

az: ( 2.546e-10, 0, 5.62) Å

Metric tensor:

gij: ( 31.58, 1.431e-09, 1.431e-09) Å

( 1.431e-09, 31.58, 6.48e-20) Å

( 1.431e-09, 6.48e-20, 31.58) Å

Volume: 177.5 Å<sup>3</sup>

Unit cell volume: 177.5 Å<sup>3</sup>

Reciprocal lattice parameters: [0.17793594306049823, 0.17793594306049823, 0.17793594306049823, 90.00000000257246, 90.00000000516778, 90.00000000516778]

Reciprocal lattice vectors:

Rax: ( 0.1779, -8.06e-12, -8.06e-12) Å<sup>-1</sup>

Ray: ( 0, 0.1779, 0) Å<sup>-1</sup>

Raz: ( 0, 0, 0.1779) Å<sup>-1</sup>

Reciprocal lattice metric tensor:

Rgij: ( 0.03166, -1.422e-12, -1.422e-12) Å<sup>-1</sup>

(-1.422e-12, 0.03166, 6.382e-23) Å<sup>-1</sup>

(-1.422e-12, 6.382e-23, 0.03166) Å<sup>-1</sup>

Reciprocal unit cell volume: 0.005634 Å<sup>-3</sup>

hkl range: 7 7 7

Diffraction angle, d, h, k, l:

2Q= 15.75 d= 5.62 ( -1 0 0)

2Q= 15.75 d= 5.62 ( 0 -1 0)

(cut)

2Q= 22.35 d= 3.97394 ( -1 -1 0)

2Q= 22.35 d= 3.97394 ( -1 0 -1)

2Q= 22.35 d= 3.97394 ( 1 0 1)

¥

# Madelung potential

**Sum of Coulomb potential in 3D is very slowly converging**

Potential is proportional to  $r^{-1}$

Polarization potential due to +/- ions is to  $r^{-2}$

Number of ions on the sphere surface at radius  $r$  is to  $r^2$

=> Contribution of ions from a surface region at  $r$   
to Coulomb sum is almost constant, independent of  $r$

$$U_{ij}(r_{ij}) = \frac{Z_i Z_j e^2}{4\pi\epsilon_0} \frac{1}{r_{ij}} + U_{Rij}(r_{ij})$$

$$U = \frac{1}{2} \sum_{i \neq j} U_{ij} = -A_M N_A \frac{Z^2 e^2}{4\pi\epsilon_0 R} + U_R$$

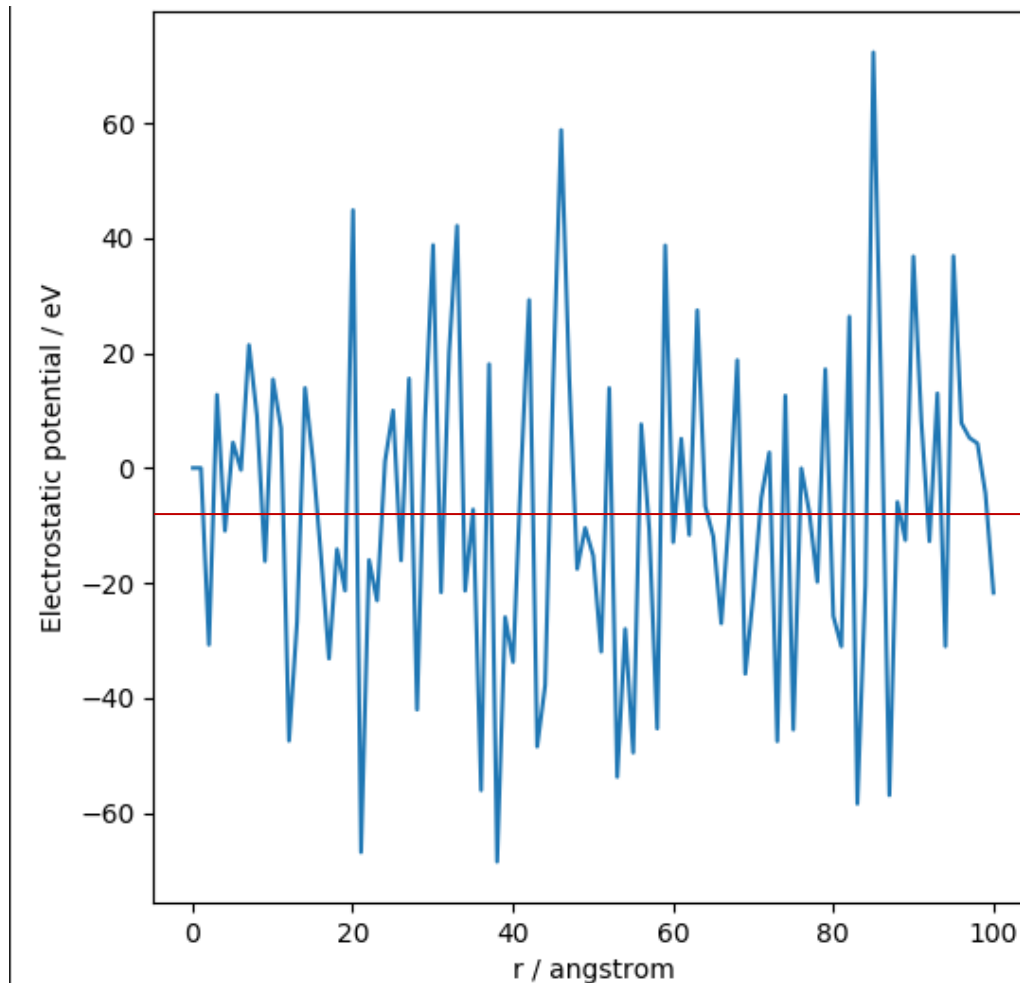
$$A_M = \frac{1}{2} \sum_{i \neq j} \frac{1}{r_{ij} / R} \quad \text{Madelung constant}$$

Crystal structure	$A_r$
Rock salt type (NaCl)	1.7476
CsCl type (CsCl)	1.7627
Zinc blend (CuCl)	1.6380
Wurzite (ZnO)	1.6413
Cu <sub>2</sub> O type	4.116
Fluorite type (CaF <sub>2</sub> )	2.520

# Madelung potential: Simple sum

python crystal\_MP\_simple.py

Coulomb sum in sphere with the radius  $r$



**Exact: -8.9 eV**

Rock salt type

y=11.961

# Efficient Coulomb sum: Evjen method

Sum up Coulomb potential in units with zero net charge

Ion charges:  $Z_i$

On boundary plane :  $1/2Z_i$

On boundary edge :  $1/4Z_i$

On boundary corner :  $1/8Z_i$

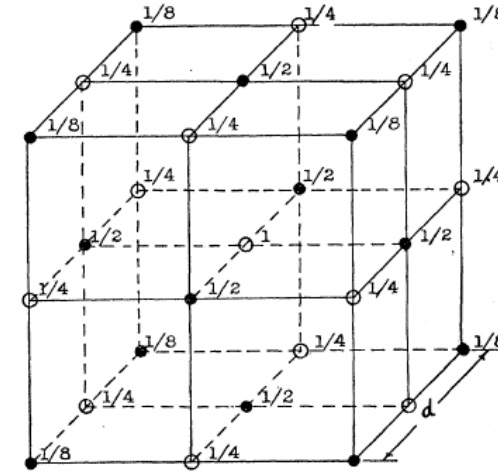


Fig. 1. Elementary cell of the NaCl-type.

Madelung constant of Rock salt type structure

$$A_M = -\frac{1}{2} \sum_{n_x, n_y, n_z = -\infty, \neq (0,0,0)}^{\infty} (-1)^{n_x + n_y + n_z} \frac{1}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$$

$$A_M = 6 \times \frac{1}{2} \times \frac{1}{\sqrt{1}} - 12 \times \frac{1}{4} \times \frac{1}{\sqrt{1+1}} + 8 \times \frac{1}{8} \times \frac{1}{\sqrt{1+1+1}} = 1.456$$

# Madelung potential: Evjen method

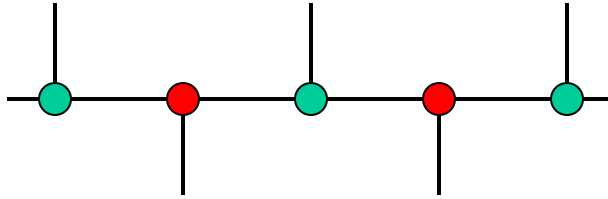
Usage: `python crystal_MP_Evjen.py ncell`

n <sub>cell</sub>	MP	Madelung constant
1	-8.9766	1.7517691
2	-8.95586	1.7477211
3	-8.95521	1.7475955
4	-8.9511	1.7475744
5	-8.95508	1.7475686
6	-8.95507	1.7475665
8	-8.95506	1.7475652
10	-8.95506	1.7475648
Exact (精確值)		1.74756

Rock salt type

# 3D sum of Coulomb potential: Ewald method

Periodic calculation can be enhanced by FT?



Periodic positions of charge

=> converted to the origin of FT data

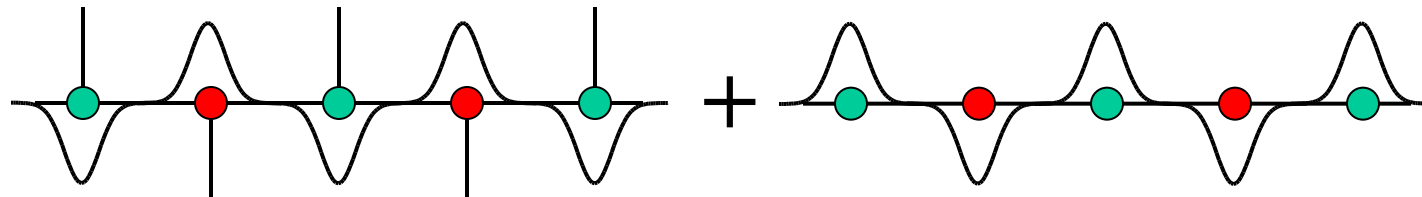
But the charges are point charges

=> converted to infinite in FT space

---

---

=> Calculate for charges with finite width  
(拡がりのある電荷の周期配列として計算する)



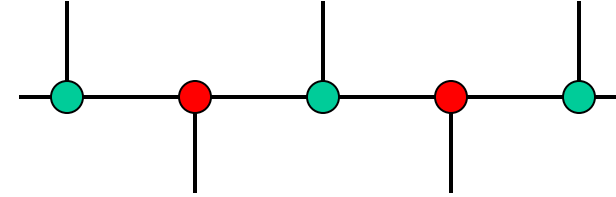


# 3D sum of Coulomb potential: Ewald method

The finite width charge distributions are converted by FT

=> Take faster calculation parts in the real space and the reciprocal space  
 拡がった電荷のフーリエ変換を利用し、実空間和と逆空間和の計算の速い部分をとる

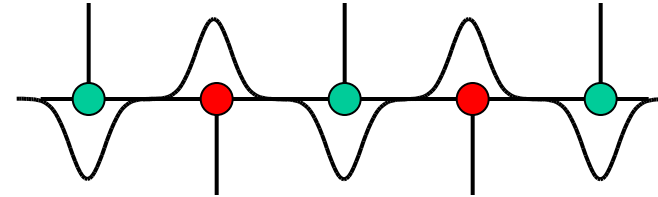
$$\Phi_i = K_C Z_i \sum_j \frac{Z_j}{r_{ij}} \quad (K_C = \frac{e^2}{4\pi\epsilon_0})$$



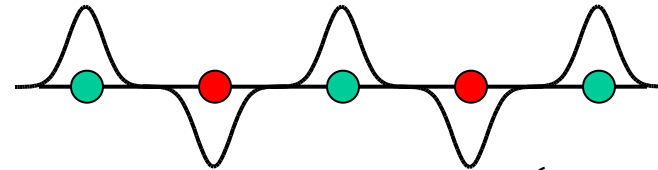
---

---


$$\Phi_i^I = K_C Z_i \sum_j Z_j \frac{\text{erfc}(\alpha|r_{ij}|)}{|r_{ij}|}$$



$$\Phi_i^{II} = K_C \frac{Z_i}{\pi V} \sum_{h,k,l} \frac{1}{|\mathbf{G}_{hkl}|^2} \exp\left(-\frac{\pi^2 |\mathbf{G}_{hkl}|^2}{\alpha^2}\right) \times \left\{ \cos(2\pi \mathbf{G}_{hkl} \cdot \mathbf{r}_i) \sum_j Z_j \cos(2\pi \mathbf{G}_{hkl} \cdot \mathbf{r}_j) + \sin(2\pi \mathbf{G}_{hkl} \cdot \mathbf{r}_i) \sum_j Z_j \sin(2\pi \mathbf{G}_{hkl} \cdot \mathbf{r}_j) \right\}$$



$$\mathbf{G}_{hkl} \cdot \mathbf{r}_i = hx_i + ky_i + lz_i$$

$$\Phi_i^{III} = K_C Z_i \frac{2\alpha Z_i}{\sqrt{\pi}}$$

$$\Phi_i = \Phi_i^I + \Phi_i^{II} - \Phi_i^{III}$$

# Madelung potential: Ewald method

Usage: `python crystal_MP_Ewald.py alpha prec`

Alpha	Precision	MP	Madelung constant	Range	Time (s)
0.3	$10^{-3}$	-8.95558	1.7476663	10.1/222 0.063 /222	0.016/0 /0.016
<b>0.3</b>	<b><math>10^{-5}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	11.9/333 0.105 /222	0.031/0 /0.031
<b>0.3</b>	<b><math>10^{-7}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	13.6/333 0.147 /333	0.047/0 /0.047
0.2	$10^{-3}$	-8.95506	1.7475646	15.2/333 0.028 /111	0.042/0 /0.042
0.6	$10^{-3}$	-8.95607	1.7477629	5.1/111 0.25 /333	0 /0.016 /0.016
0.8	$10^{-3}$	-8.95584	1.747718	3.8/111 0.45 /444	0 /0.016 /0.016
<b>0.2</b>	<b><math>10^{-10}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	24.3/555 0.093/222	0.16/0 /0.16
<b>0.4</b>	<b><math>10^{-10}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	12.1/333 0.373/444	0.036/0.016/0.052
<b>0.5</b>	<b><math>10^{-10}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	9.7/222 0.58 /555	0.016/0.016/ <b>0.031</b>
<b>0.6</b>	<b><math>10^{-10}</math></b>	<b>-8.95506</b>	<b>1.7475646</b>	8.1/222 0.84 /666	0.016/0.031/0.047
Exact (精確値)			1.74756		

Range:  $R_{\max} [\text{\AA}]/n_{x\max}n_{y\max}n_{z\max}$   $G_{\max} [\text{\AA}^{-1}]/h_{\max}k_{\max}l_{\max}$   
 Time: Real space sum / Reciprocal space sum / Total [s]

Rock salt type

# Comparison: Evjen method

Rock salt type

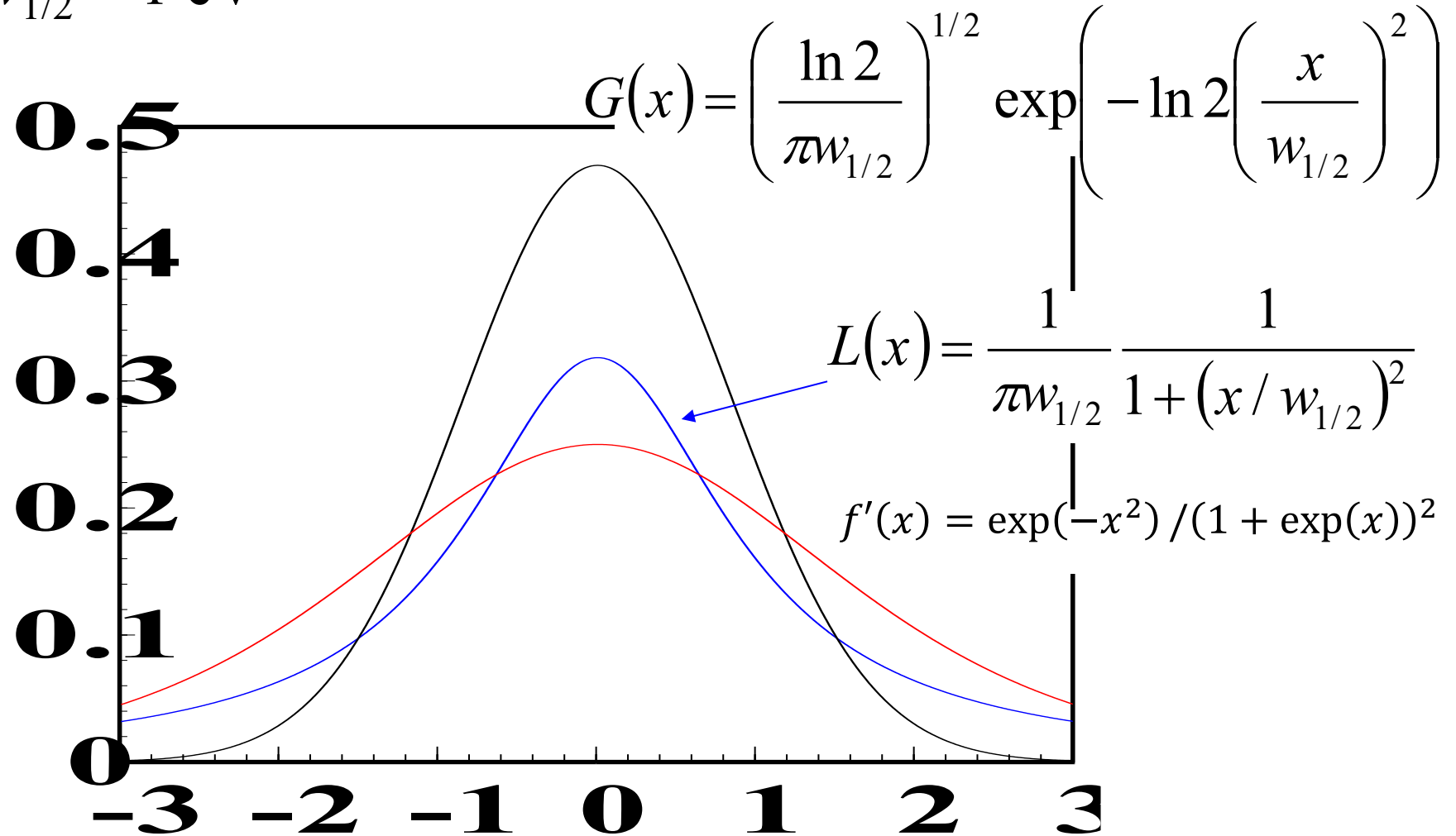
$$A_M = -\frac{1}{2} \sum_{n_x, n_y, n_z = -\infty, \neq (0,0,0)}^{\infty} (-1)^{n_x+n_y+n_z} \frac{1}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$$

nx	ny	nz	r	m	Z	S(mZ/r)	f	S(mZf/r)	nx	ny	nz	r	m	Z	S(mZ/r)	f	S(mZf/r)
0	0	1	1	6	-1	-6	0.5	-3	0	0	1	1	6	-1	-6	1	-6
0	1	1	1.4142	12	1	8.48528	0.25	2.12132034	0	1	1	1.4142	12	1	8.48528	1	8.485281374
1	1	1	1.7321	8	-1	-4.6188	0.13	-0.5773503	1	1	1	1.7321	8	-1	-4.6188	1	-4.61880215
						<b>-2.13</b>		<b>-1.456</b>	0	0	2	2	6	1	3	1	3
									0	1	2	2.2361	24	-1	-10.733	1	-10.7331263
									0	2	2	2.8284	12	1	4.24264	1	4.242640687
nx	ny	nz	r	m	Z	S(mZ/r)	f	S(mZf/r)	1	1	2	2.4495	24	1	9.79796	1	9.797958971
0	0	1	1	6	-1	-6	1	-6	1	2	2	3	24	-1	-8	1	-8
0	1	1	1.4142	12	1	8.48528	1	8.48528137	2	2	2	3.4641	8	1	2.3094	1	2.309401077
1	1	1	1.7321	8	-1	-4.6188	1	-4.6188022	0	0	3	3	6	-1	-2	0.5	-1
0	0	2	2	6	1	3	0.5	1.5	0	1	3	3.1623	24	1	7.58947	0.5	3.794733192
0	1	2	2.2361	24	-1	-10.733	0.5	-5.3665631	0	2	3	3.6056	24	-1	-6.6564	0.5	-3.32820118
0	2	2	2.8284	12	1	4.24264	0.25	1.06066017	0	3	3	4.2426	12	1	2.82843	0.25	0.707106781
1	1	2	2.4495	24	1	9.79796	0.5	4.89897949	1	1	3	3.3166	24	-1	-7.2363	0.5	-3.61813613
1	2	2	3	24	-1	-8	0.25	-2	1	2	3	3.7417	48	1	12.8285	0.5	6.414269806
2	2	2	3.4641	8	1	2.3094	0.13	0.28867513	1	3	3	4.3589	24	-1	-5.506	0.25	-1.3764944
						<b>-1.52</b>		<b>-1.7518</b>	2	2	3	4.1231	24	-1	-5.8209	0.5	-2.9104275
									2	3	3	4.6904	24	1	5.11682	0.25	1.279204298
									3	3	3	5.1962	8	-1	-1.5396	0.13	-0.19245009
															<b>-1.91</b>		<b>-1.7470</b>

Exact value = **1.7476**

# A: Smearing functions

$$w_{1/2} = 1 \text{ eV}$$



# A: Methfessel Paxton function

M. Methfessel and A.T. Paxton, High-precision sampling  
for Brillouin-zone integration in metals, Phys. Rev. B **40** (1989) 3616

**Expand the delta function with Hermitian polynomials**

$$\delta(x) = \sum_{n=0}^{\infty} A_n H_{2n}(x) \exp(-x^2) \quad A_n = \frac{(-1)^n}{n! 4^n \sqrt{\pi}}$$

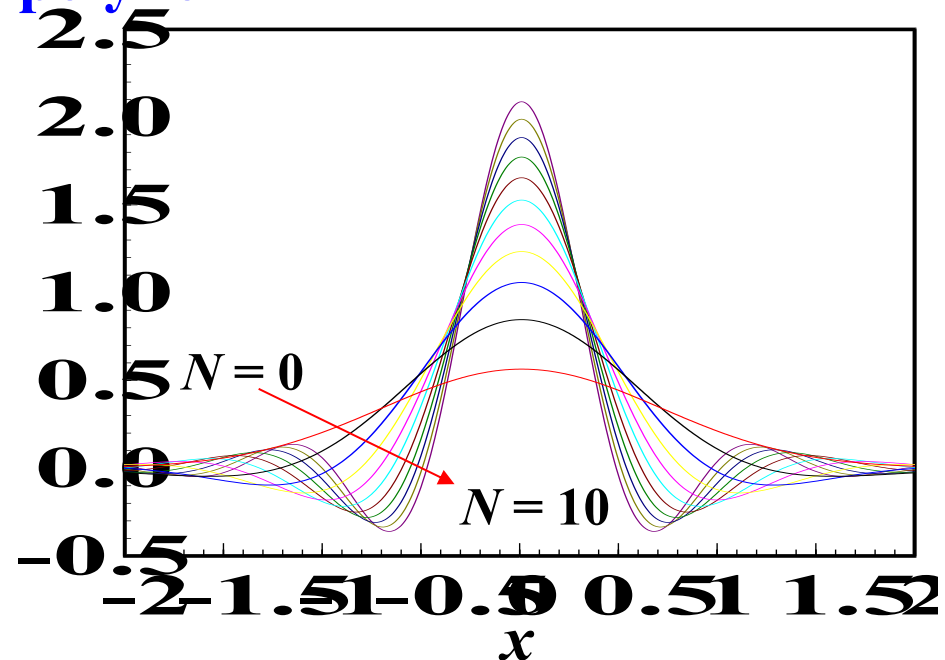
$$D_N(x) = \sum_{n=0}^{2N} A_n H_{2n}(x) \exp(-x^2)$$

**$D_N(x)$  is a  $(2N+1)$ -order polynomial,  
orthogonal to a  $2N$  or less order polynomial**

**Approximation of the Step Function**

$$S_N(x) = 1 - \int_{-\infty}^x D_N(t) dt$$

$$S_0(x) = (1/2)(1 - \operatorname{erf}(x))$$



# Hermitian polynomial

$$\left(\frac{d^2}{dx^2} - 2x \frac{d}{dx} + 2n\right) H_n(x) = 0 \quad \text{solution of a problem}$$

$$H_n(x) = n! \sum_{m=0}^{\text{int}(n/2)} \frac{(-1)^m}{m! (n-2m)!} (2x)^{n-2m}$$

$$H_0(x) = 1, H_1(x) = 2x$$

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$$

$$H_n'(x) = 2nH_{n-1}(x) = 2xH_n(x) - H_{n+1}(x)$$

**$H_n(x)\exp(-x^2/2)$  is an orthonormal basis**

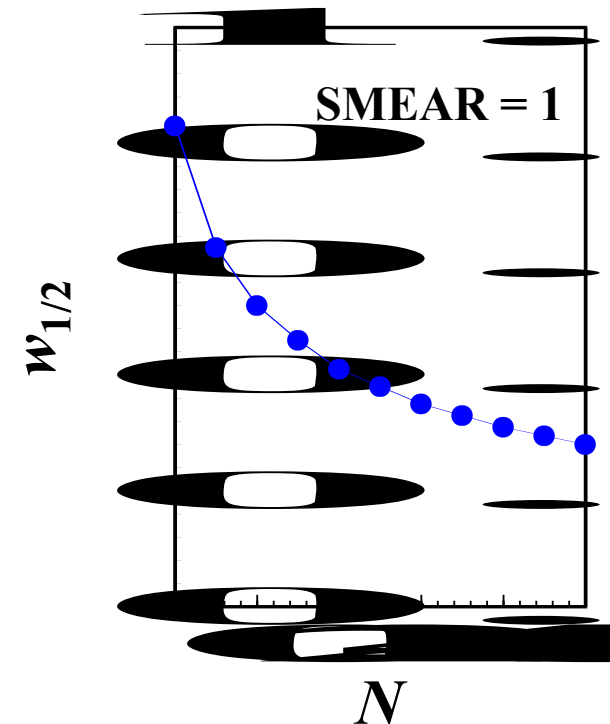
$$\int_{-\infty}^{\infty} H_n(x) H_m(x) \exp(-x^2) dx = \delta_{mn} 2^n \sqrt{\pi} n!$$

**Wavefunction of harmonic oscillator model:**

$$\Psi_n(x) = (2^n \sqrt{\pi} n!)^{1/2} H_n(x) \exp(-x^2/2)$$

# A: Characteristics of Methfessel Paxton Functions

1. If the band structure and DOS can be approximated by polynomials of the  $2N$ -th order or less, smearing with an  $N$ -th-order MP function will **not produce an integration error**.
2. In the case of a simple band structure or DOS, **integration error will be zero** even if the SMEAR width is quite large.
3. When the band structure is complex (e.g., d-system), the optimal SMEAR width is comparable to Gaussian
4. The actual smearing width  $w_{1/2}$  depends on the value of the order  $N$  and SMEAR width.
5. **Negative values or values over 1.0 for occupancy**



# A: Smearing of density of states $D(E) * f_s(E)$

$$D(E) = D_{V0}(E_V - E)^{1/2} + D_{C0}(E - E_C)^{1/2} * (1 + \text{rand}[-0.5, 0.5])$$

Smearing:  $D(E) * f_s(E) = \int D(E') f_s(E' - E) dE'$   
with  $w = 0.2$  eV

