

Computational Materials Science (計算材料学特論)

Lecture materials updated (this morning, 8:24)

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

2024年度Q2 計算材料学特論 (資料: 英語 + 日本語版)

Computational Materials Science 2024 Q2

数値解析に関する講義資料・pythonプログラム (神谷担当分)

Lecture materials for numerical analyses (by Kamiya)

講義で使うプレゼン資料は

共通して使うpythonプログラム」の下にあります

Lecture presentation slides are found after the "Common python programs" section below.

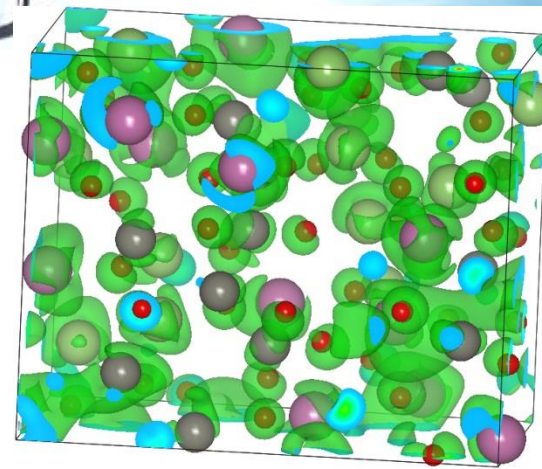
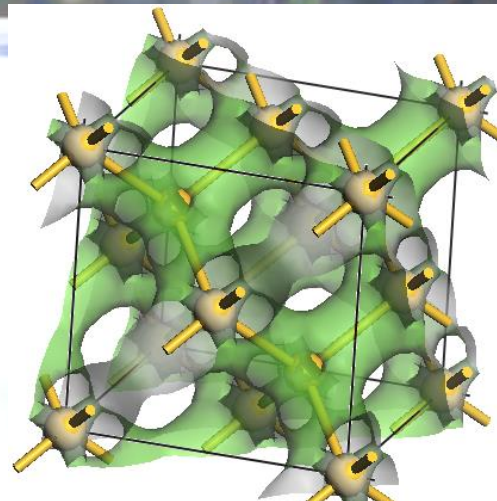
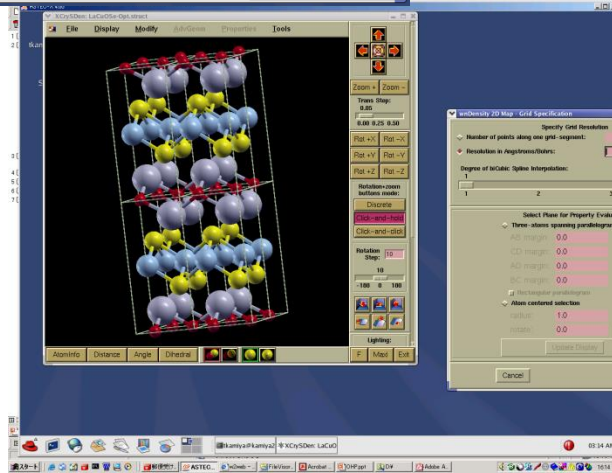
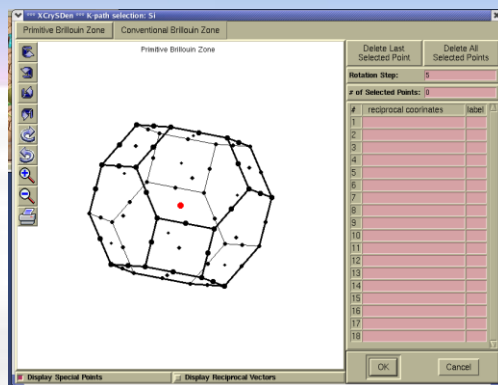
Update News:

- June 28, 8:24 Lecture materials on June 28 have been updated ([20240628FT_Matrix2.zip](#))
- June 27, 10:09 Lecture materials on June 28 have been uploaded ([20240628FT_Matrix.zip](#))
- June 27, 9:25 Lecture materials on June 25 have been updated ([20240627LSQEquationOptimize.zip](#))
- June 25, 12:51 Lecture materials on June 25 have been updated
- June 24, 10:40 Lecture materials on June 25 have been updated
- June 23, 10:12 Lecture materials on June 25 have been uploaded
- June 23, 9:32 Lecture materials on June 21 have been updated ([20240621InterporlateSmoothing.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://conf.msl.titech.ac.jp/Lecture/>

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

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

Evaluation (Kamiya)

- **Small quiz**
Not evaluate correctness of the answers
but consider how you answered them
- **Term-end assignment**
Problems will be given at the end of Q2
from T2SCHOLAR

Explanation of the answers, June 28

課題解答の解説

PROBLEM, June 28

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

PROBLEM: Answer can be in Japanese or English

- (i) Find (x, y) to minimize $\exp(-x^2) * \sin(x+y)$ by your-chosen algorithms
hint: use SD method with fixed alpha
if df/dx and df/dy are approximated, use central differences
- (i) Optional: Propose if you have any other numerical analysis you want to learn in Computational Materials Science
- (ii) Optional: Propose if you have any python program (should be simple) you want to learn

PROBLEM, June 28

PROBLEM: Answer can be in Japanese or English

(i) Find (x, y) to minimize $\exp(-x^2) * \sin(x+y)$ by your-chosen algorithms

hint: use SD method with fixed alpha

if df/dx and df/dy are approximated, use central differences

See [minimize2D_answer.xlsx](#)

$\exp(-x^2) * \sin(x+y)$ is a multi-valued functional.

general solutions:

$x = 0$ and $y = (-1/2 + 2n)\pi$ where n is integer

PROBLEM, June 25

- (i) **Optional: Propose if you have any other numerical analysis you want to learn in Computational Materials Science**
- (ii) **Optional: Propose if you have any python program (should be simple) you want to learn**

- 最適化手法全般について前回の授業以上に学びたい。おすすめの書籍を知りたい
- 数値解析を勉強する上でのおすすめの参考書

Textbooks

- 自分で何かしらのデータをフィッティングするプログラムを書く上での注意点とおすすめの方法 (scipy optimize?)
データ例: XRDピーク、光の干渉によるフリンジ、エネルギーバンド

How to make fitting program

- 測定データを解析するためにベイズ最適化を用いたプログラムをpythonのライブラリのoptunaを用いて作成していますが、

Bayesian optimization

- tetrahedron法について(基礎理論と使うメリット、どのような場面で利用されるか)

3D numerical integration

- モンテカルロ法に関連する(特にising modelなど) 事

Monte Carlo method: for integration, stochastic simulations, optimization

- 回帰で用いられる正則化について(基礎的な内容)

Machine learning regression / sparse modelling / parameter reduction

Will be explained on July 5th

PROBLEM, June 25: Regression methods

See (Japanese) tutorial course:

<http://conf.msl.titech.ac.jp/D2MatE/2022Tutorial/tutorial2022.html>

材料計算科学・データ解析チュートリアルコース 2022年度

2022年度チュートリアルコースは終了いたしました。

チュートリアル資料の更新は、下記の4/2 14:04更新と録画公開で終了しました。

配布プログラムの更新は、[Topページ](#)にて行います。

-
- 2023/4/2 14:04 第5回チュートリアル (3月22日) 講義資料 最終更新版: [20230322Tutorial.zip](#)
2023/4/1 注: Arrhenius plot機能について、活性化エネルギーの計算に間違いがあったものを修正しました。
 - 2023/3/22 第1～5回チュートリアル録画 2023年3月いっぱいので予定で公開: [2022年度チュートリアル録画](#)

チュートリアル資料 再構成版: チュートリアルの資料をテーマごとにまとめなおしました。

ファイル名が日本語になっています。ダウンロードができない場合、tkamiya@msl.titech.ac.jp までご連絡

- [01-注意.pdf](#)
- [02-Launcher.pdf](#)
- [02-python-tips.pdf](#)
- [03-強化学習プログラムbayes_gp_gui.pyの使い方.pdf](#)
- [04-線形最適化・最小二乗法\(回帰\).pdf](#)
- [05-機械学習\(回帰\).pdf](#)
- [06-非線形最適化・最小二乗法.pdf](#)
- [07-強化学習\(ガウス過程回帰ベイズ推定\)の基礎.pdf](#)
おまけ: [07'-ベイズ統計.pdf](#)
- [08-スペクトル解析.pdf](#)

材料計算科学・データ解析チュートリアルコース 2022年度

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 D2MatE](#) 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。

Algorithms available with python scikit-learn

Linear regression:

多重線形回帰

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

多項式回帰

```
from sklearn.preprocessing import PolynomialFeatures
model = PolynomialFeatures(degree = 2, interaction_only = False, include_bias = True, order = 'C')
```

Ridge回帰

```
from sklearn.linear_model import Ridge
model = Ridge(alpha = 0.05)
```

LASSO回帰

```
from sklearn.linear_model import Lasso
model = Lasso(alpha = 0.05)
```

Elastic Net回帰

```
from sklearn.linear_model import ElasticNet
model = ElasticNet(alpha = 0.05)
```

Non-parametric regression:

Kernel Ridge回帰

```
from sklearn.kernel_ridge import KernelRidge
model = KernelRidge(alpha = 1.0, kernel = 'rbf')
```

Gauss過程回帰

```
from sklearn.gaussian_process import GaussianProcessRegressor
model = GaussianProcessRegressor()
```

多層パーセプトロン (多層ニューラルネットワーク)

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(max_iter = 1000, hidden_layer_sizes = (10,), activation = 'logistic', solver = 'sgd',
                      learning_rate_init = 0.01)
```

Algorithms available with python scikit-learn

分類器系 (Regression by classifier):

Random Forest回帰

```
from sklearn.ensemble import RandomForestRegressor  
model = RandomForestRegressor()
```

勾配ブースティング木 回帰

```
from sklearn.ensemble import GradientBoostingRegressor  
model = GradientBoostingRegressor()
```

サポートベクターマシーン 回帰

```
from sklearn.svm import SVR  
model = SVR(kernel='linear', C=1, epsilon=0.1, gamma='auto')
```

Algorithms available with python scikit-learn

<https://qiita.com/futakuchi0117/items/72ce4afae9adcccd6e18>

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet, SGDRegressor
from sklearn.linear_model import PassiveAggressiveRegressor, ARDRegression, RidgeCV
from sklearn.linear_model import TheilSenRegressor, RANSACRegressor, HuberRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor, HistGradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor, GradientBoostingRegressor, VotingRegressor, StackingRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.cross_decomposition import PLSRegression

reg_dict = {"LinearRegression": LinearRegression(),
            "Ridge": Ridge(),
            "Lasso": Lasso(),
            "ElasticNet": ElasticNet(),
            "Polynomial_deg2": Pipeline([('poly', PolynomialFeatures(degree=2)),('linear', LinearRegression())]),
            "Polynomial_deg3": Pipeline([('poly', PolynomialFeatures(degree=3)),('linear', LinearRegression())]),
            "Polynomial_deg4": Pipeline([('poly', PolynomialFeatures(degree=4)),('linear', LinearRegression())]),
            "Polynomial_deg5": Pipeline([('poly', PolynomialFeatures(degree=5)),('linear', LinearRegression())]),
            "KNeighborsRegressor": KNeighborsRegressor(n_neighbors=3),
            "DecisionTreeRegressor": DecisionTreeRegressor(),
            "RandomForestRegressor": RandomForestRegressor(),
            "SVR": SVR(kernel='rbf', C=1e3, gamma=0.1, epsilon=0.1),
            "GaussianProcessRegressor": GaussianProcessRegressor(),
            "SGDRegressor": SGDRegressor(),
            "MLPRegressor": MLPRegressor(hidden_layer_sizes=(10,10), max_iter=100, early_stopping=True, n_iter_no_change=5),
            "ExtraTreesRegressor": ExtraTreesRegressor(n_estimators=100),
            "PLSRegression": PLSRegression(n_components=10),
            "PassiveAggressiveRegressor": PassiveAggressiveRegressor(max_iter=100, tol=1e-3),
            "TheilSenRegressor": TheilSenRegressor(random_state=0),
            "RANSACRegressor": RANSACRegressor(random_state=0),
            "HistGradientBoostingRegressor": HistGradientBoostingRegressor(),
            "AdaBoostRegressor": AdaBoostRegressor(random_state=0, n_estimators=100),
            "BaggingRegressor": BaggingRegressor(base_estimator=SVR(), n_estimators=10),
            "GradientBoostingRegressor": GradientBoostingRegressor(random_state=0),
            "VotingRegressor": VotingRegressor([('lr', LinearRegression()), ('rf', RandomForestRegressor(n_estimators=10))]),
            "StackingRegressor": StackingRegressor(estimators=[('lr', RidgeCV()), ('svr', LinearSVR())], final_estimator=RandomForestRegressor(n_estimators=10)),
            "ARDRegression": ARDRegression(),
            "HuberRegressor": HuberRegressor(),
            }
```

A simple case: Ridge regression

Add L2 norm penalty for coefficients \mathbf{a}_k to suppress over learning

Fitting function: $f(x) = \sum_{k=0}^p a_k x^{(k)}$

Objective function $S = \sum_{j=1}^n (\sum_{k=1}^p a_k x_j^{(k)} - y_j)^2 + \alpha \sum_{k=0}^p a_k^2 \quad \lambda \geq 0$

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^n x_j^{(k')} (\sum_{k=1}^p a_k x_j^{(k)} - y_j) + 2\alpha a_{k'} = 0$$

$$\sum_{k=1}^p a_k \sum_{j=1}^n x_j^{(k')} x_j^{(k)} + \alpha a_{k'} = \sum_{k=1}^p \sum_{j=1}^n x_j^{(k')} x_j^{(k)} y_j$$

$$\begin{pmatrix} \sum x^{(1)} x^{(1)} + \alpha & \sum x^{(1)} x^{(2)} & \sum x^{(1)} x^{(3)} & \dots & \sum x^{(1)} x^{(p)} \\ \sum x^{(2)} x^{(1)} & \sum x^{(2)} x^{(2)} + \alpha & \sum x^{(2)} x^{(3)} & & \sum x^{(2)} x^{(p)} \\ \sum x^{(3)} x^{(1)} & \sum x^{(3)} x^{(2)} & \sum x^{(3)} x^{(3)} + \alpha & & \sum x^{(3)} x^{(p)} \\ \vdots & & & \ddots & \\ \sum x^{(p)} x^{(1)} & \sum x^{(p)} x^{(2)} & \sum x^{(p)} x^{(3)} & & \sum x^{(p)} x^{(p)} + \alpha \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum x^{(1)} y_i \\ \sum x^{(2)} y_i \\ \sum x^{(3)} y_i \\ \vdots \\ \sum x^{(p)} y_i \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^n x_j^{(k)} x_j^{(k')} + \alpha \delta_{kk'} = \mathbf{x}^{(k)} \cdot \mathbf{x}^{(k')} + \alpha I_p$$

Solve $\mathbf{X}\mathbf{A}=\mathbf{Y}$ and find best α (regularization parameter)

PROBLEM, June 28

- (i) **Optional: Propose if you have any other numerical analysis you want to learn in Computational Materials Science**
- (ii) **Optional: Propose if you have any python program (should be simple) you want to learn**

How to perform Boltzmann transport properties calculations, like Seebeck coefficient and electronic conductivity. I want to learn python code for calculating Boltzmann transport properties.

DFT code + **BoltzTraP2**

BoltzTraP2 is provided by PyPI (see <https://pypi.org/project/BoltzTraP2/>),
So you can install with pip on Linux.

Install URL: <https://boltztrap2y.readthedocs.io/en/latest/Installation.html>

1. Change your python environmnt
2. % pip install wheel
% pip install BoltzTraP2

The executable (launcher) of BoltzTraP2 is bt2.

Check if you can run bt2.

% bt2

Equations of simplified Boltzmann equation

Read <https://www.sciencedirect.com/science/article/pii/S0010465518301632> for exact math.

$$\tau(E, T) = \tau_0 T^p (E - E_C)^{r-1/2} \quad r: \text{scattering factor}$$

$$\langle \tau^k \rangle = -\frac{2}{3} \int_{E_C}^{\infty} (E - E_C) \tau(E)^k D_C(E - E_C) \frac{\partial f_e(E)}{\partial E} dE / n_e$$

$$\text{Carrier density } n_e = \int_{E_C}^{\infty} D_C(E) f_e(E) dE = \sum_{\text{occupied states in CB}} n_i$$

$$\text{Conductivity } \sigma_x = e n_e \frac{e}{m_e^*} \langle \tau^1 \rangle \quad \text{Drift mobility } \mu_{\text{drift}} = \frac{e}{m_e^*} \langle \tau^1 \rangle$$

$$\text{Hall coefficient: } R_H = \frac{E_y}{J_x B_z} = \frac{F_H}{qn} = \frac{1}{qn_H} \quad (q = -e \text{ for electron, } +e \text{ for hole})$$

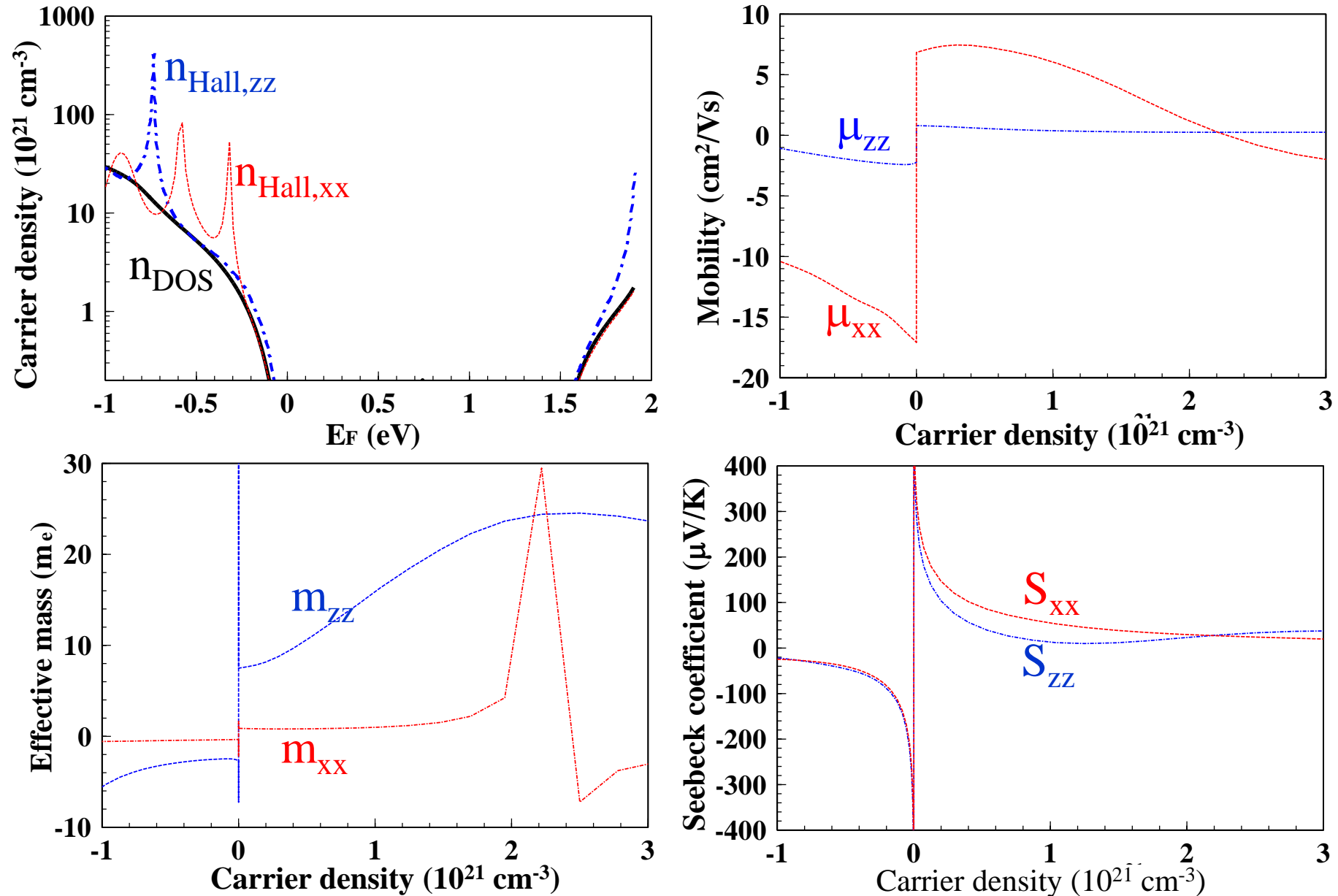
$$\text{Hall factor: } F_H = \frac{\langle \tau^2 \rangle}{\langle \tau \rangle^2} \quad \text{Hall mobility: } \mu_H = F_H \mu_{\text{drift}} \quad \text{Hall carrier density: } n_H = \frac{1}{F_H} n_e$$

$$\text{Seebeck coefficient } S = -\frac{k}{e} \frac{\int \left(-\frac{\partial f}{\partial E} \right) D(E) v^2 \tau \left[\frac{E - E_F}{kT} \right] dE}{\int \left(-\frac{\partial f}{\partial E} \right) D(E) v^2 \tau dE} + \frac{1}{e} \frac{\partial E_F}{\partial T}$$

Equations of tensors: <https://www.sciencedirect.com/science/article/pii/S0010465506001305>

With constant relaxation time approximation,
assume $\tau(E) = \text{constant}$

Carrier transport properties of LaCuOSe by BoltzTraP



PROBLEM, June 28

- (i) **Optional: Propose if you have any other numerical analysis you want to learn in Computational Materials Science**
- (ii) **Optional: Propose if you have any python program (should be simple) you want to learn**

仕事関数の評価法として密度汎関数を用いる手法を知りたい (Calculation of work function by DFT)

See e.g., https://www.vasp.at/wiki/index.php/Computing_the_work_function

Procedure to calculate work function

1. Make a surface supercell model (slab model)
2. Perform DFT for ideal (base) crystal model and the surface model
3. Align the valence band edges (E_{VBM}) of the ideal and the surface models, by aligning average core potential, core level eigen energies etc.
4. Take valence band maximum from the ideal crystal model
5. Estimate vacuum level (E_{vac}) from the surface model

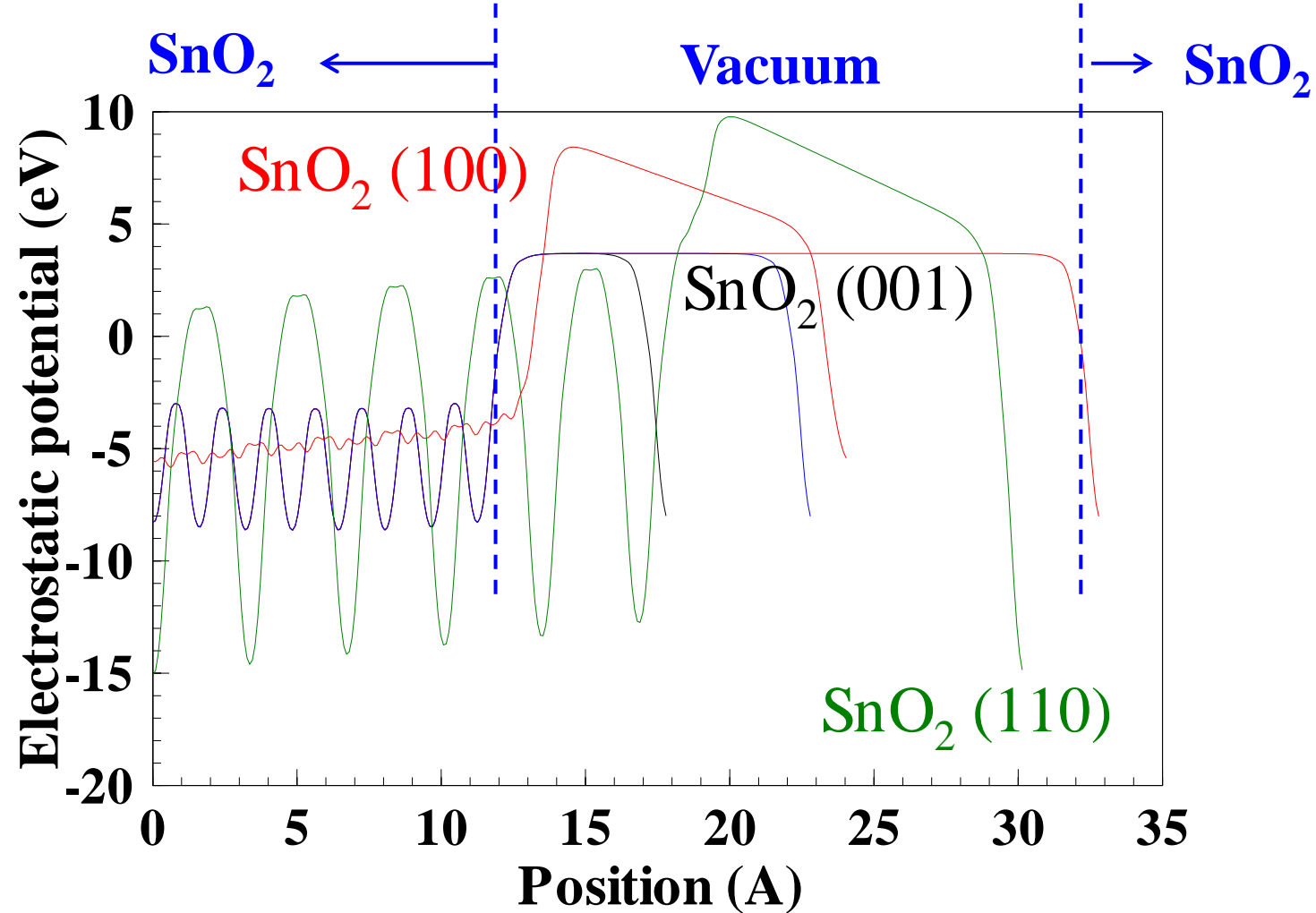
Note that DFT calculation cannot provide reliable E_{vac} due to the ‘DFT’s band gap problem’.

Calculate E_{vac} from charge density using the Poisson’s equation.

6. Then you can obtain ionization potential by $\text{IP} = E_{\text{vac}} - E_{\text{VBM}}$.

Work function is obtained by the Fermi level E_{F} , $\text{WF} = E_{\text{vac}} - E_{\text{F}}$.

Work function of SnO₂ by VASP



$$E_{\text{e-pot,vac}} = 3.68 \text{ eV}$$

$$E_{\text{tot,bulk}}(\text{N-e}) - E_{\text{tot,bulk}}(\text{N}) = 4.36 \text{ eV}$$

$$\text{WF} = [E_{\text{e-pot,vac}} + E_{\text{tot,bulk}}(\text{N-e})] - E_{\text{tot,bulk}}(\text{N}) = 8.04 \text{ eV (Obs: 8.6 eV)}$$

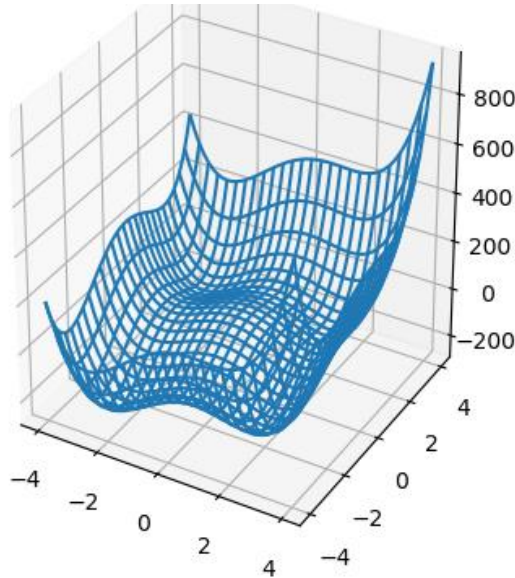
Non-linear (NL) optimization

非線形最適化

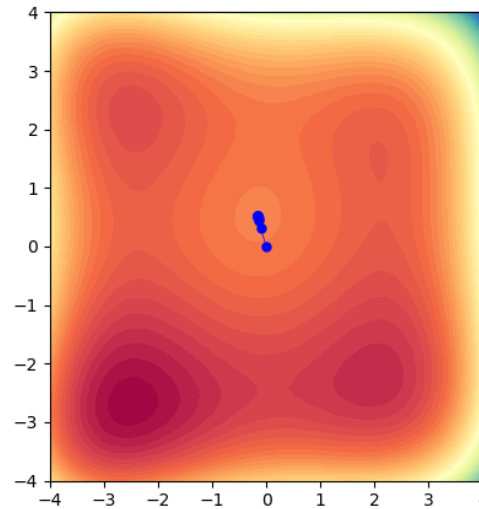
Optimize.py for Simplex method (not working without tklib)

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

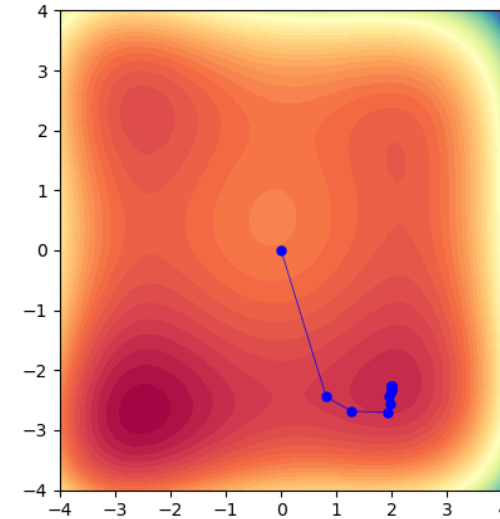
e.g.: python optimize.py 0.0 1.0 simplex



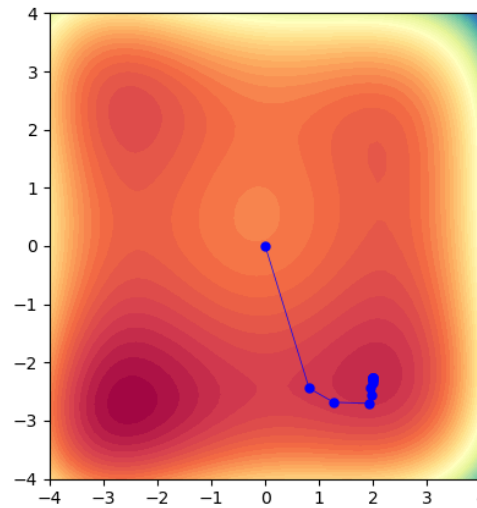
From (0.0 0.0) Newton



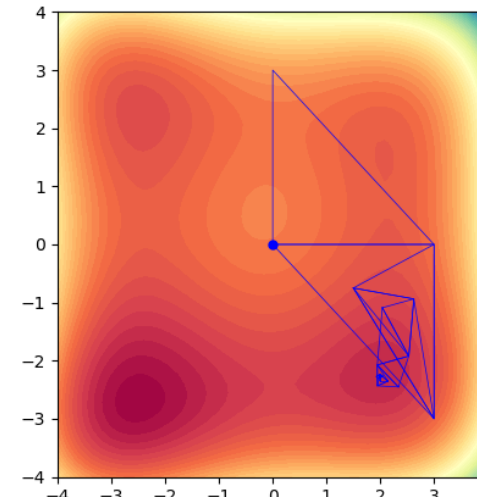
From (-1.0 -1.0) DFP golden



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

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()`

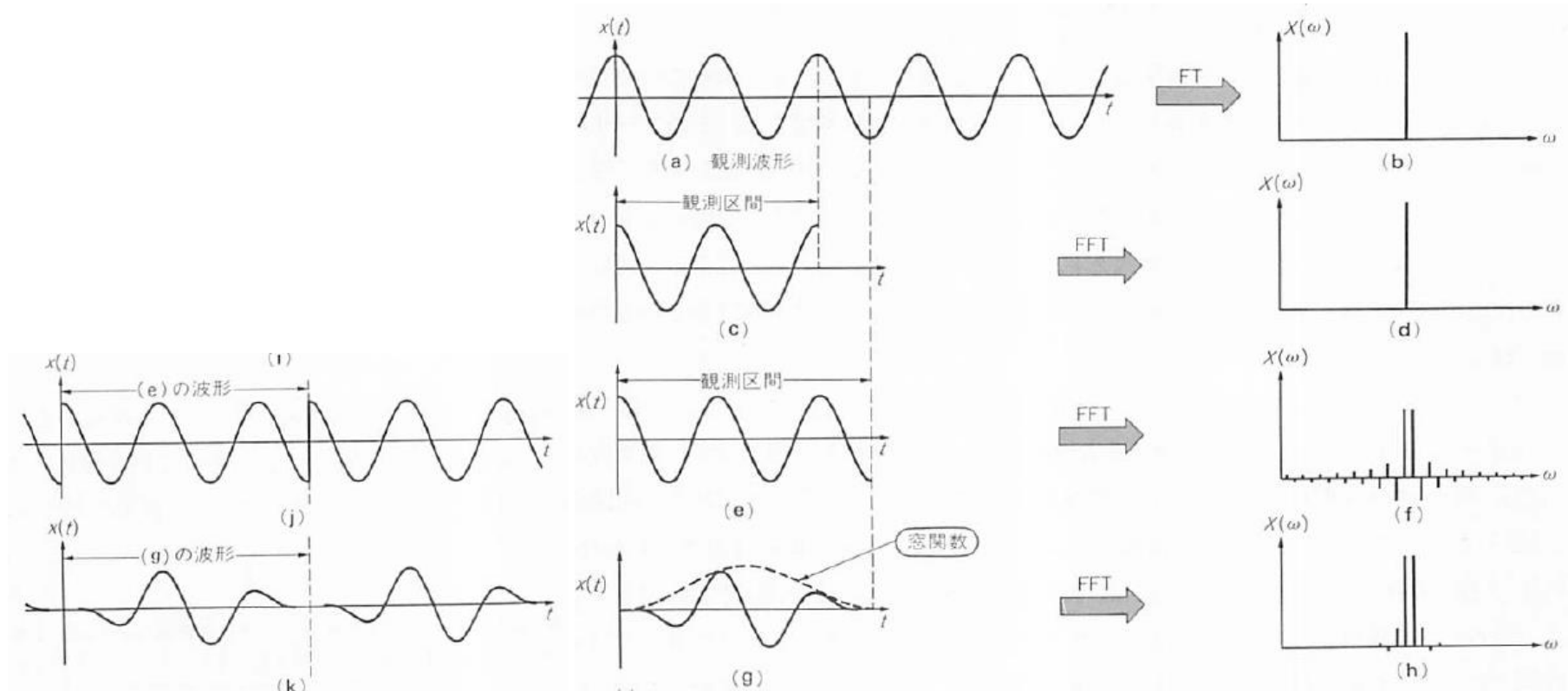
Time for DFT/FFT (sec)

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 ∞ , 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

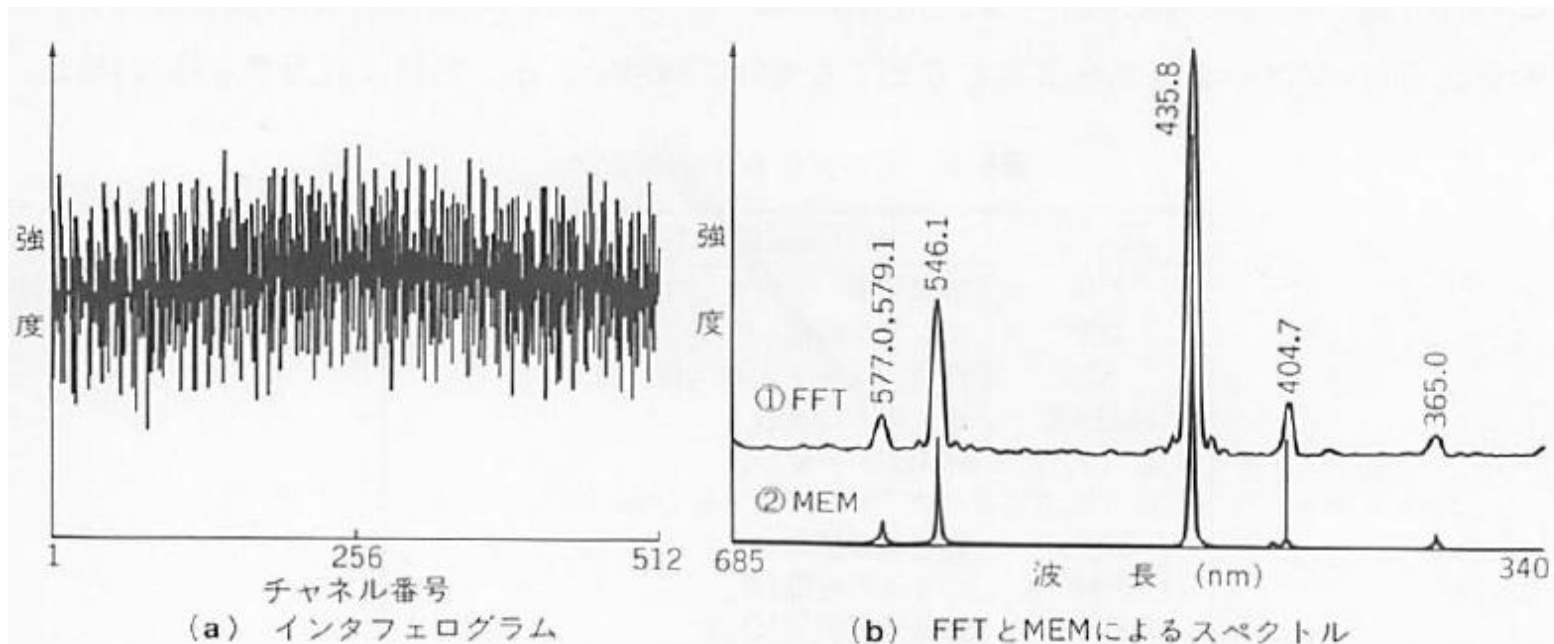
- Assume the lost data would have some constraints
- Use the concept of 'information entropy' and maximize it to estimate the spectrum
- Akaike's autoregressive model (赤池による自己回帰モデル) => identical to MEM

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



MEM-Rietveld analysis

坂田誠 日本結晶学会誌 30, 135 (1988)

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

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

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

=> smoothing ρ' and suppress fringes and ghost peaks

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

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 \mathbf{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

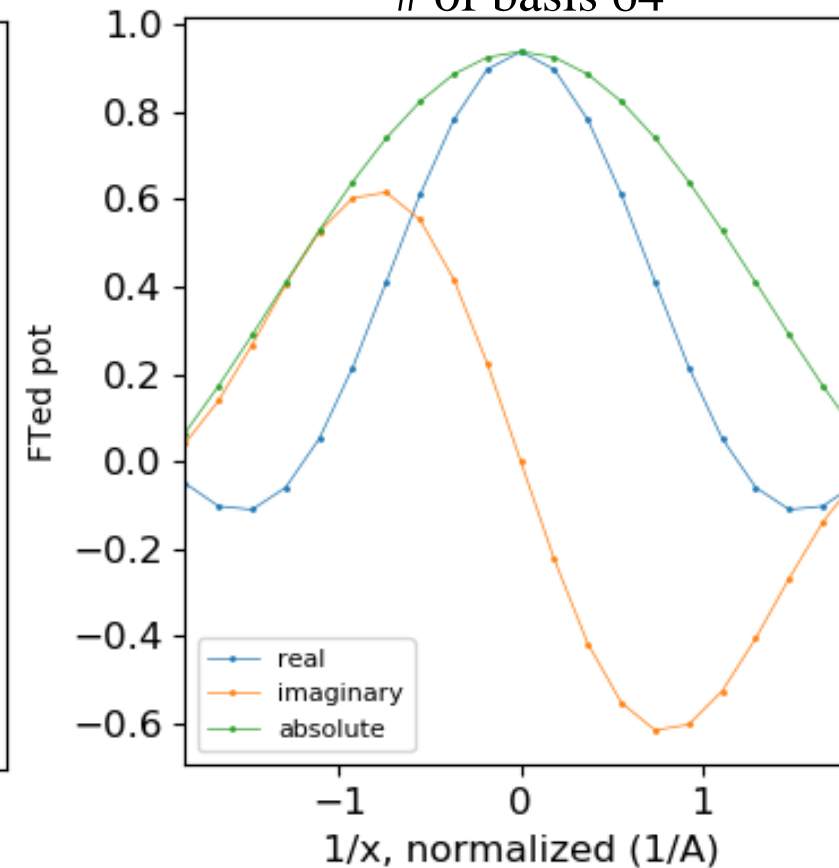
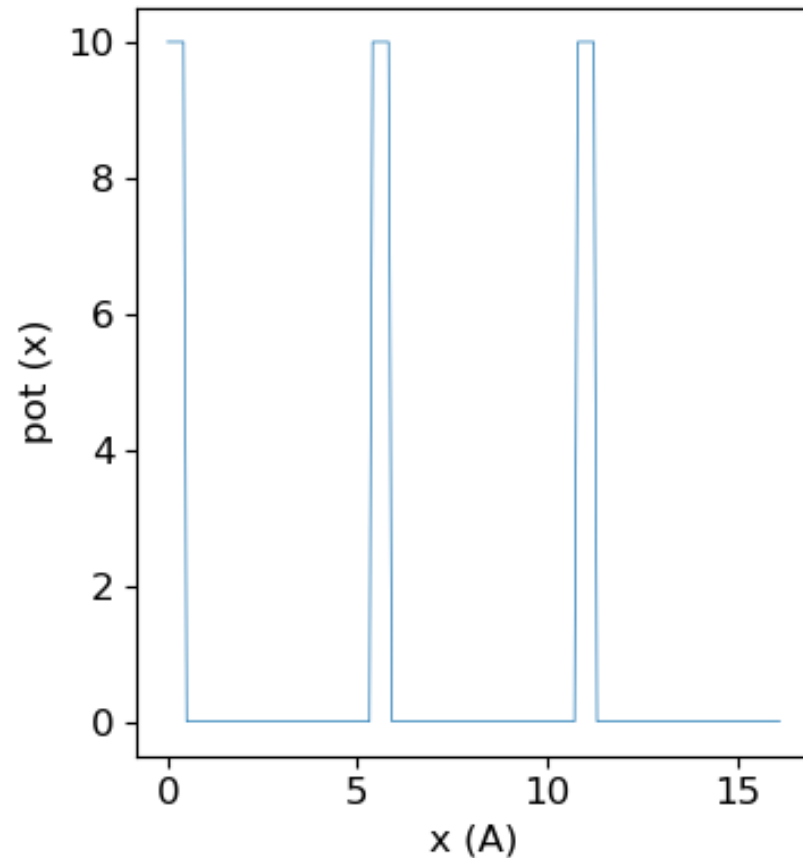
Lattice parameter (Si) $a = 5.4064 \text{ \AA}$ $m^* = 1.0m_e$

Potential $V(x)$: barrier width 0.5 \AA barrier height 10.0 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
```

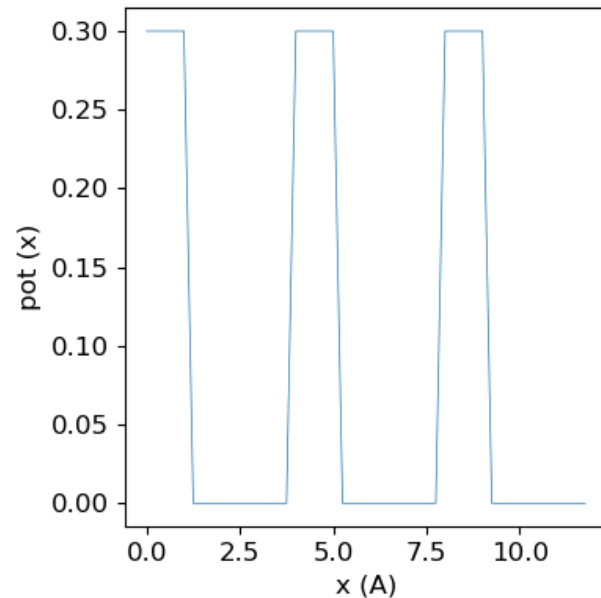
```
python pw1d.py band 5.4064 64 rect 0.5 10.0 3 -0.5 0.5 21
```

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

$a = 4.0 \text{ \AA}$

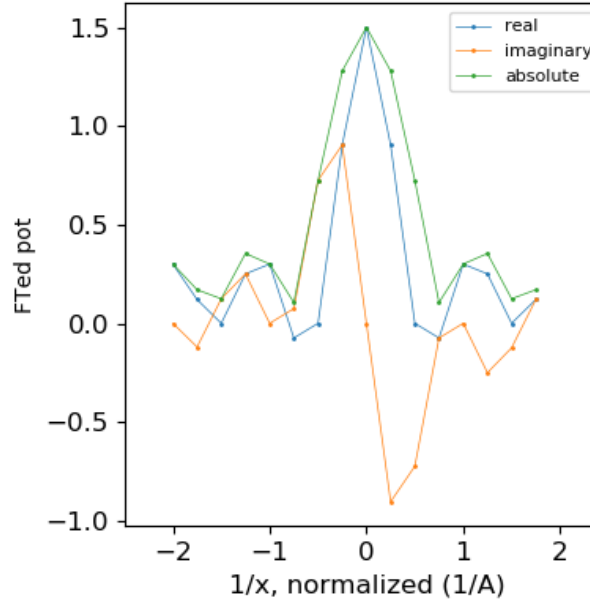
potential $V(x)$:

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



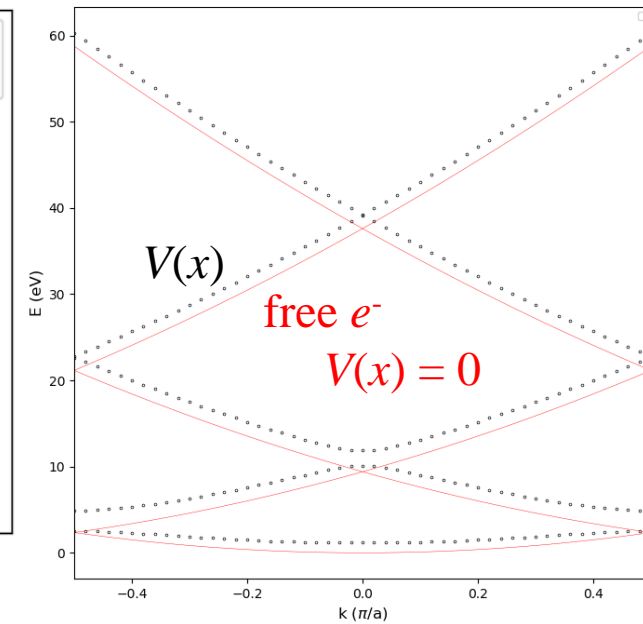
FT coefficients of potential

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)

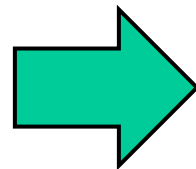
Diagonalization of $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$

=> 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 $U =$

$$U = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & & & & & \vdots \\ \vdots & & \cos \theta & & -\sin \theta & & \vdots \\ \vdots & & & 1 & & & \vdots \\ \vdots & & \sin \theta & & \cos \theta & & \vdots \\ \vdots & & & & & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

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
 \Rightarrow Solve eigen values of A_R

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

Applications

応用

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>

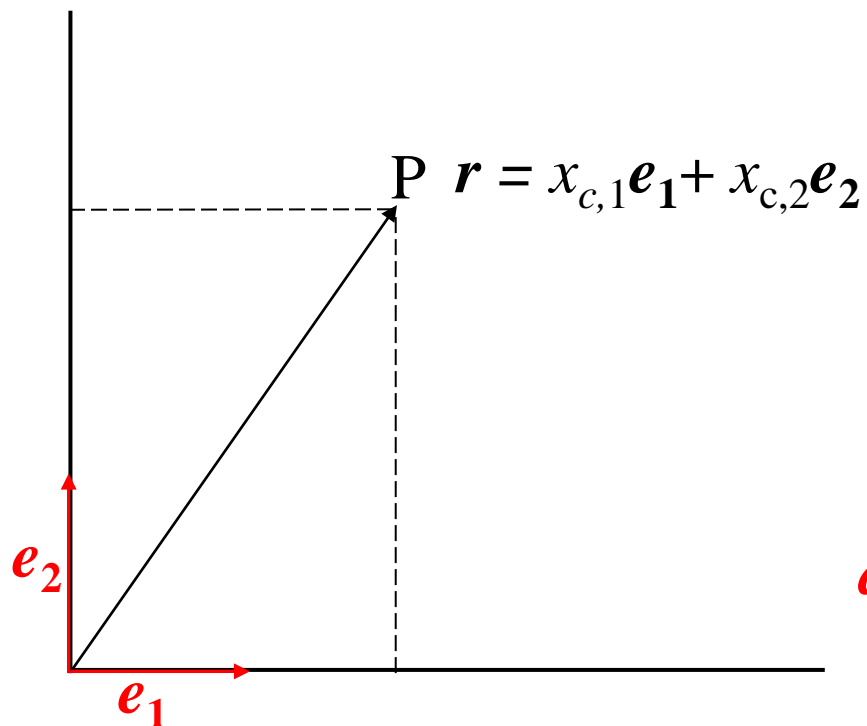
Python: numpy.linalg, scipy.linalg

matrix.py

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

一般座標系 (general coordinate system)

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

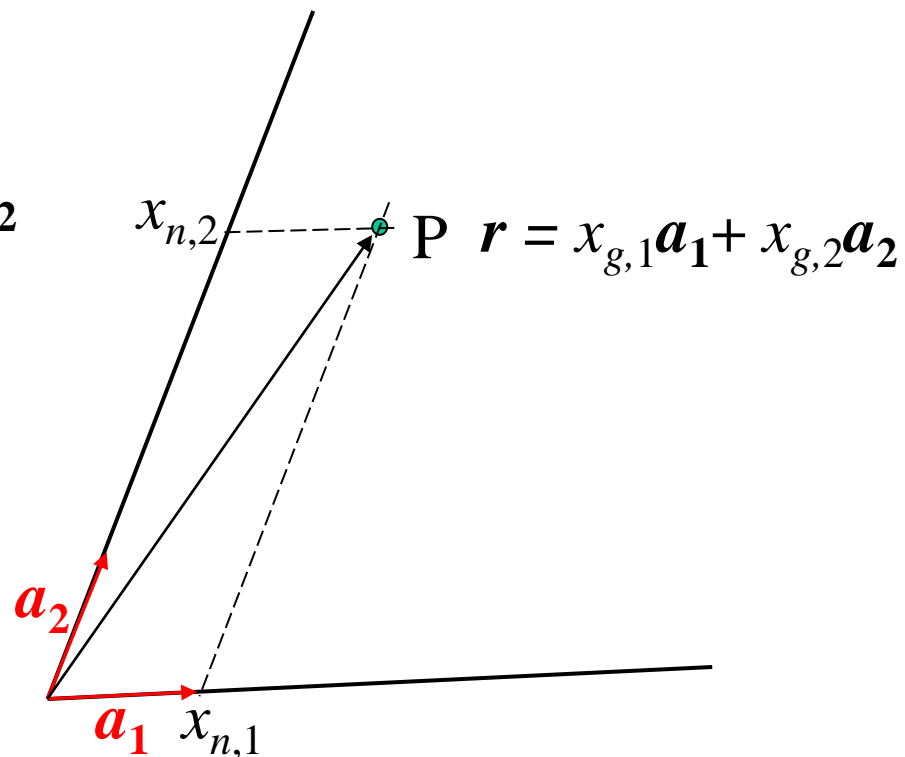


正規直交系 (orthonormal system)

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

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

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,

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

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

Fractional coordinates in crystal

(結晶の内部座標)

Lattice parameters: a, b, c ($= a_1, a_2, a_3$), α, β, γ ($= \alpha_{23}, \alpha_{13}, \alpha_{12}$)

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()

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

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

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

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

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

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

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 A³

Unit cell volume: 177.5 A³

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⁻¹

Ray: (0, 0.1779, 0) A⁻¹

Raz: (0, 0, 0.1779) A⁻¹

Reciprocal lattice metric tensor:

Rgij: (0.03166, -1.422e-12, -1.422e-12) A⁻¹

(-1.422e-12, 0.03166, 6.382e-23) A⁻¹

(-1.422e-12, 6.382e-23, 0.03166) A⁻¹

Reciprocal unit cell volume: 0.005634 A⁻³

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

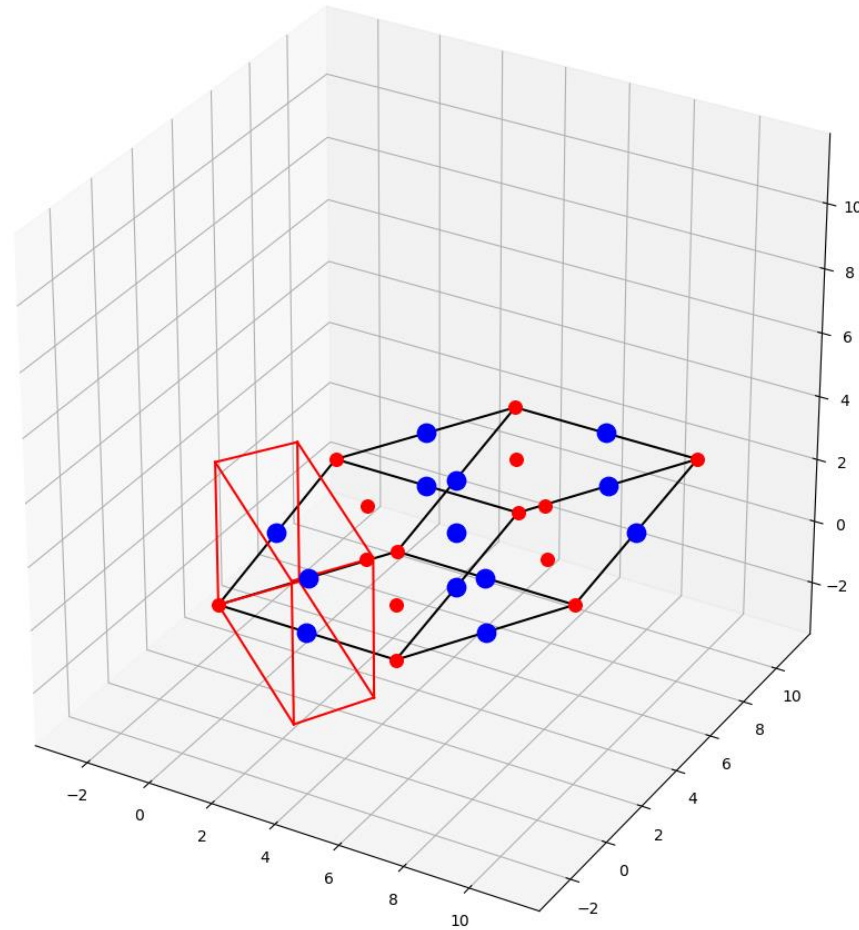
Na4 (0.5, 0.5, 0) - Na2 (0, 0.5, 0.5) + (1, 0, -1): dis = 3.974 A

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 Å³

Unit cell volume: 177.5 Å³

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⁻¹

Ray: (0, 0.1779, 0) Å⁻¹

Raz: (0, 0, 0.1779) Å⁻¹

Reciprocal lattice metric tensor:

Rgij: (0.03166, -1.422e-12, -1.422e-12) Å⁻¹

(-1.422e-12, 0.03166, 6.382e-23) Å⁻¹

(-1.422e-12, 6.382e-23, 0.03166) Å⁻¹

Reciprocal unit cell volume: 0.005634 Å⁻³

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

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

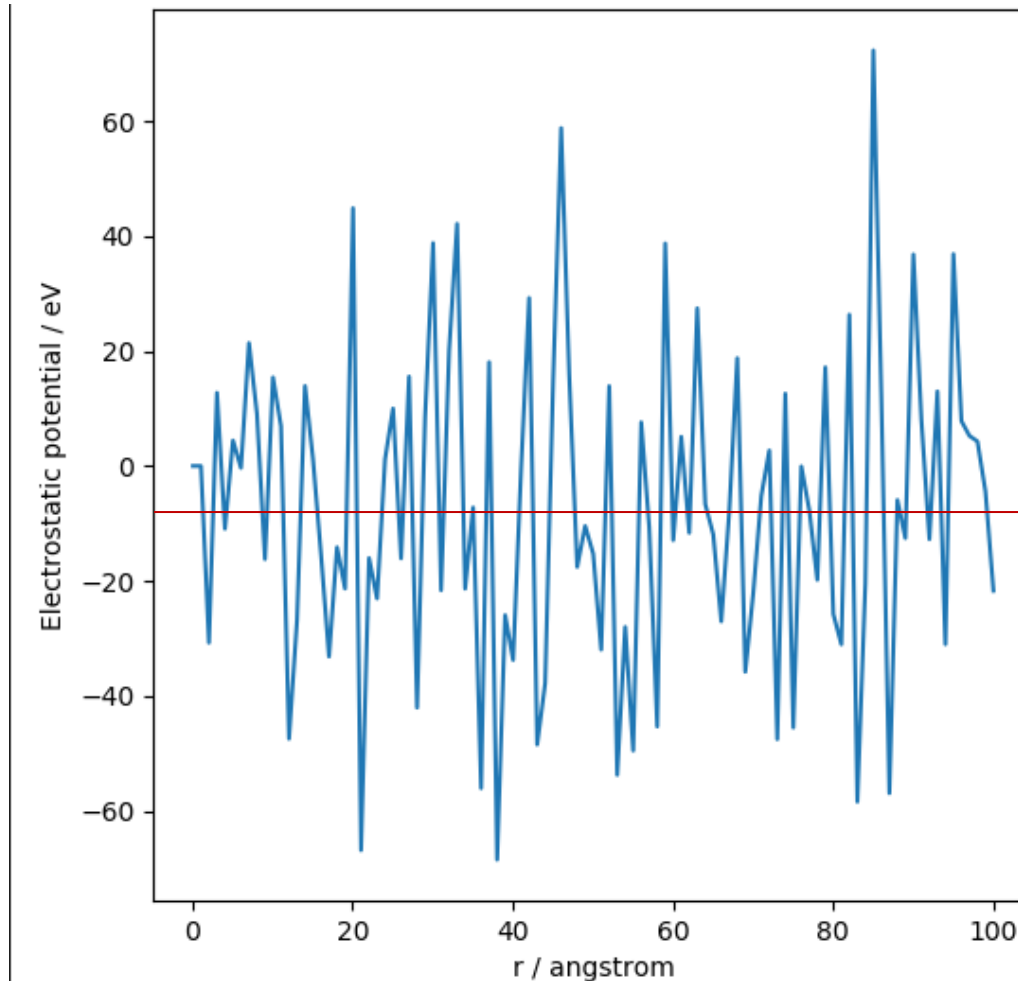
$$A_M = \frac{1}{2} \sum_{i \neq j} \frac{1}{r_{ij} / R} \quad \text{Madelung constant}$$

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 ₂ O type	4.116
Fluorite type (CaF ₂)	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 : $1/8Z_i$

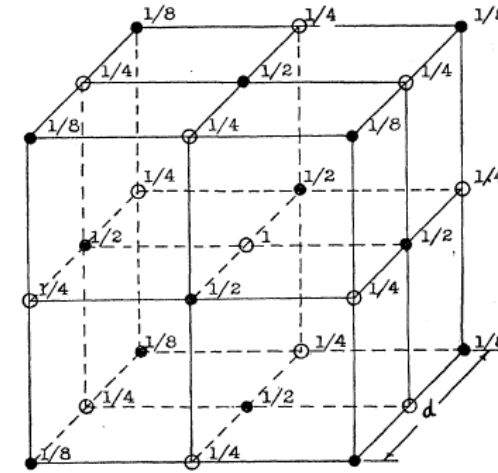


Fig. 1. Elementary cell of the NaCl-type.

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

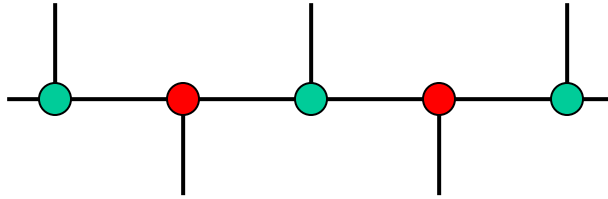
Usage: `python crystal_MP_Evjen.py ncell`

n _{cell}	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?



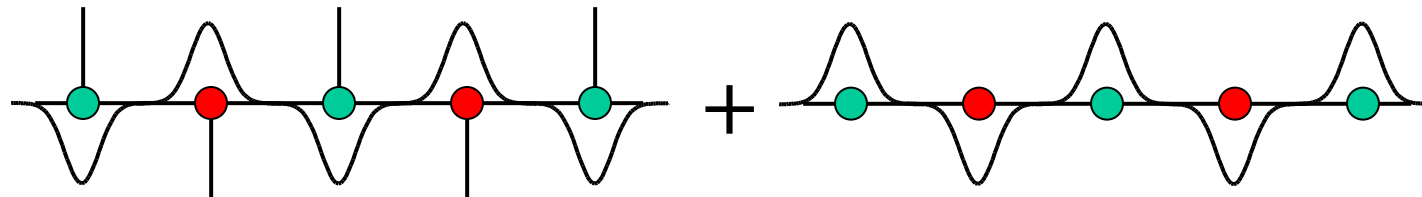
Periodic positions of charge

=> 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)

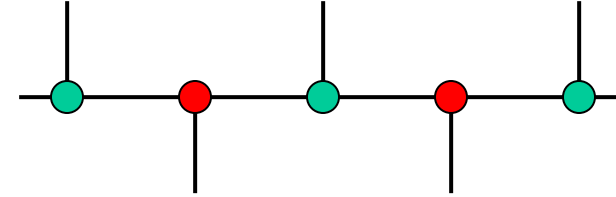


3D sum of Coulomb potential: Ewald method

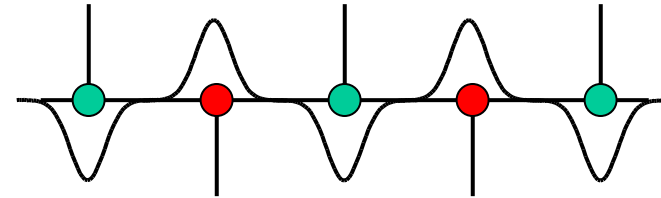
The finite width charge distributions are converted by FT

=> 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

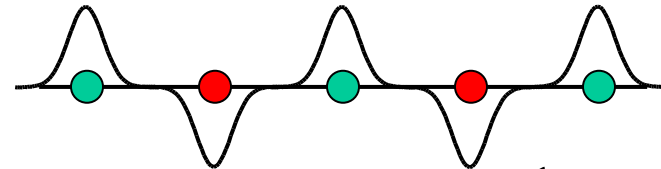
$$\Phi_i = K_C Z_i \sum_j \frac{Z_j}{r_{ij}} \quad (K_C = \frac{e^2}{4\pi\epsilon_0})$$



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



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



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

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

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

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
0.3	10^{-5}	-8.95506	1.7475646	11.9/333 0.105 /222	0.031/0 /0.031
0.3	10^{-7}	-8.95506	1.7475646	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
0.2	10^{-10}	-8.95506	1.7475646	24.3/555 0.093/222	0.16/0 /0.16
0.4	10^{-10}	-8.95506	1.7475646	12.1/333 0.373/444	0.036/0.016/0.052
0.5	10^{-10}	-8.95506	1.7475646	9.7/222 0.58 /555	0.016/0.016/ 0.031
0.6	10^{-10}	-8.95506	1.7475646	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
						-2.13		-1.456	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
						-1.52		-1.7518	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
															-1.91		-1.7470

Exact value = **1.7476**

Analytical DFT XC calculation

Transfer matrix method

図10-2 H原子の波動関数

Hartree-Fock (HF) 方程式

$$\left\{ -\frac{1}{2} \nabla^2 - \frac{Z}{r} + \int \frac{\rho(\mathbf{r}_m)}{|\mathbf{r}_m - \mathbf{r}|} d\mathbf{r}_m - \int \frac{\rho(\mathbf{r}_m)}{|\mathbf{r}_m - \mathbf{r}|} d\mathbf{r}_m \right\} \varphi(\mathbf{r}) = \varepsilon \varphi(\mathbf{r})$$

自己相互作用 (Self-interaction: SI) は HF 法では相殺される

Slater's $X\alpha$ (DFT)

$$\left\{ -\frac{1}{2} \nabla^2 - \frac{Z}{r} + \int \frac{\rho(\mathbf{r}_m)}{|\mathbf{r}_m - \mathbf{r}|} d\mathbf{r}_m - 3\alpha \left\{ \frac{3}{4\pi} \rho(\mathbf{r}) \right\}^{1/3} \right\} \varphi(\mathbf{r}) = \varepsilon \varphi(\mathbf{r})$$

DFTでは SI は相殺されず、誤差として残る

H1s-HF-LDA.py

1s軌道内の電子数 N_e を
変化

$$\alpha = 2/3$$

厳密解: $E(1s) = -13.6 \text{ eV}$

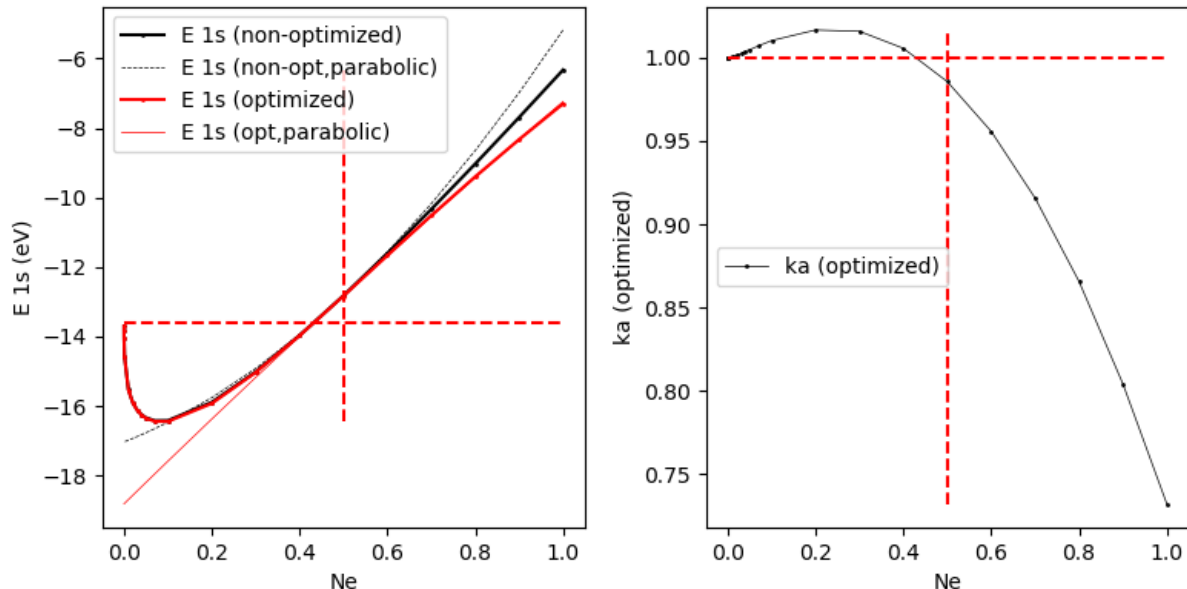


図10-2 H原子の波動関数

<http://conf.msl.titech.ac.jp/jsap-crystal/>

DFTの自己相互作用誤差: HF近似とLDAによる水素原子1s 軌道

Usage: python H1s-HF-LDA.py mode Z ka Ne

実行例1: python H1s-HF-LDA.py ng 1.0 1.0 1.0

ka = 1.0 (HFの H 1s 軌道の指数関数の係数) での
1s 軌道準位の電子数 Ne を 0 ~ 1 と変化させてプロット

実行例2: python H1s-HF-LDA.py nvg 1.0 1.0 1.0

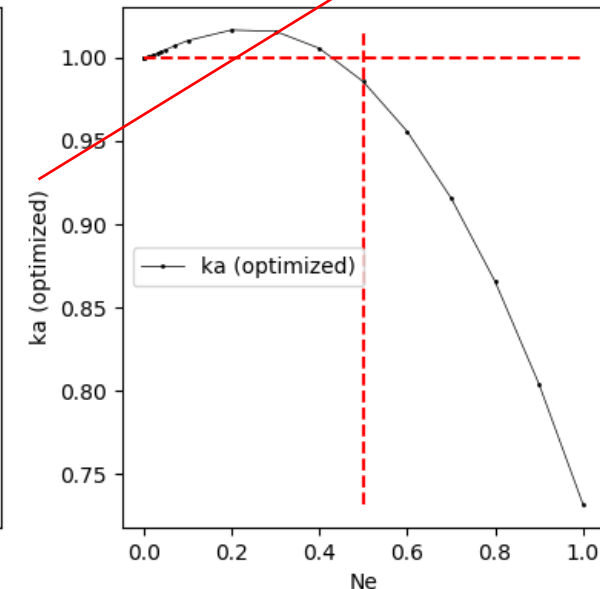
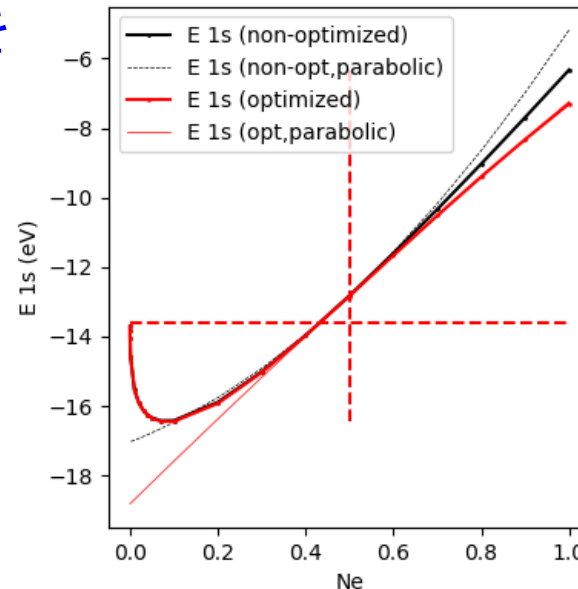
実行例1に、kaを**変分原理**で最適化させた結果を追加

python H1s-HF-LDA.py nvg 1.0 1.0 1.0

1s軌道内の電子数 N_e を
変化

厳密解: $E(1s) = -13.6 \text{ eV}$

$$R_{1s}(r) = 2a_0^{-3/2} \exp\left(-k_a \frac{1}{2} \frac{2}{a_0} r\right)$$

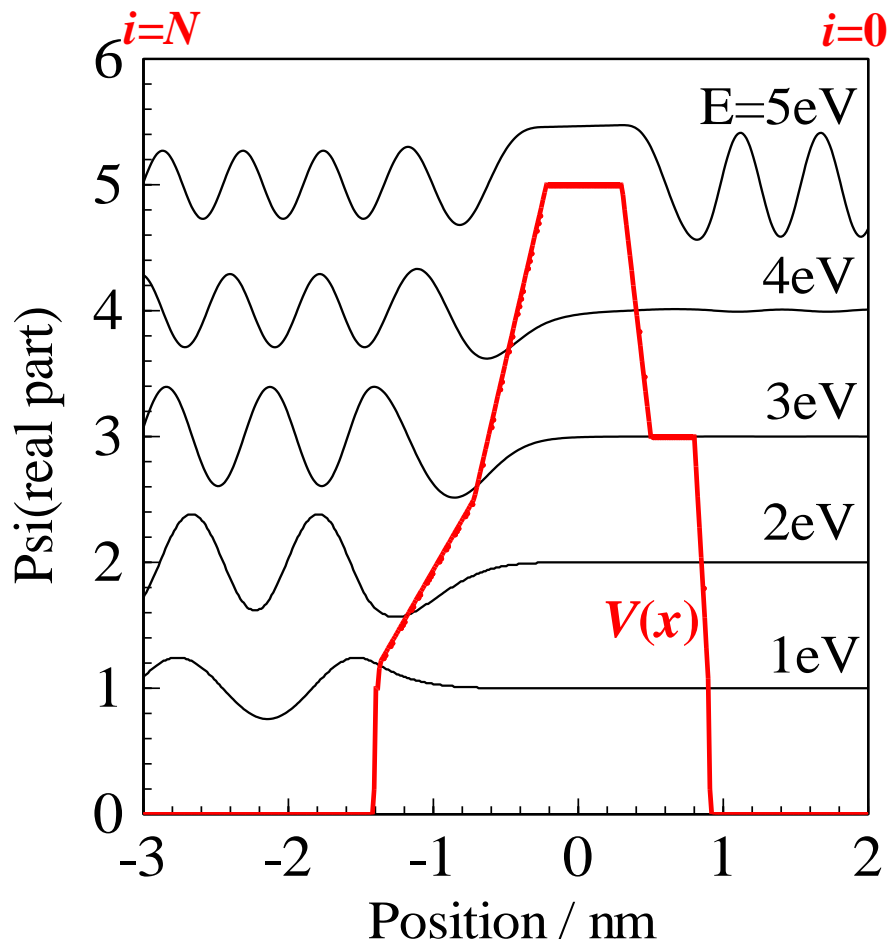


平面波近似: 転送行列法

H. Mizuta, T. Tanoue, "The Physics and Applications of Resonant Tunnelling Diodes," Cambridge Univ Press (1995)

Y. Ando and A. Itoh, J. Appl. Phys. 61 (1987) 1497

$$\Psi_i(x) = A_i \exp(ik_i x) + B_i \exp(-ik_i x) \quad k_i = \sqrt{\frac{2m_i}{\hbar^2} (E - V_i)}$$



境界条件

$$\Psi_i(x_{i+1}) = \Psi_{i+1}(x_{i+1})$$

$$m_i^{-1} \Psi'_i(x_{i+1}) = m_{i+1}^{-1} \Psi'_{i+1}(x_{i+1})$$

$$\begin{pmatrix} A_{i+1} \\ B_{i+1} \end{pmatrix} = \begin{pmatrix} \alpha^+_i P_i & \alpha^-_i / Q_i \\ \alpha^-_i Q_i & \alpha^+_i / P_i \end{pmatrix} \begin{pmatrix} A_i \\ B_i \end{pmatrix}$$

$$\alpha^\pm_i = \frac{1}{2} [1 \pm (m_{i+1} / m_i) (k_i / k_{i+1})]$$

$$P_i = \exp[i(k_i - k_{i+1})x_{i+1}]$$

$$Q_i = \exp[i(k_i + k_{i+1})x_{i+1}]$$

平面波近似: 転送行列法

H. Mizuta, T. Tanoue, "The Physics and Applications of Resonant Tunnelling Diodes," Cambridge Univ Press (1995)

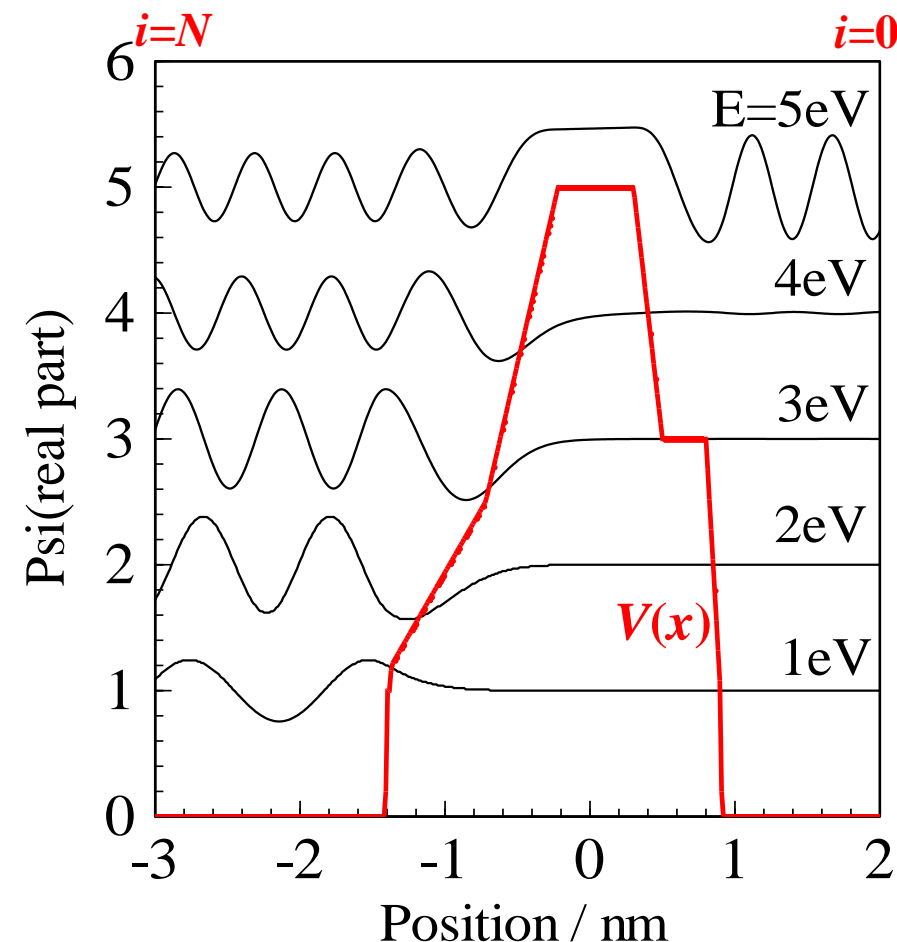
$$\begin{pmatrix} A_N \\ B_N \end{pmatrix} = \begin{pmatrix} \alpha_{N-1}^+ P_{N-1} & \alpha_{N-1}^- / Q_{N-1} \\ \alpha_{N-1}^- Q_{N-1} & \alpha_{N-1}^+ / P_{N-1} \end{pmatrix} \begin{pmatrix} A_{N-1} \\ B_{N-1} \end{pmatrix} = T_{N-1} \begin{pmatrix} A_{N-1} \\ B_{N-1} \end{pmatrix} = T_{N-1} T_{N-2} \begin{pmatrix} A_{N-2} \\ B_{N-2} \end{pmatrix} = T \begin{pmatrix} A_0 \\ B_0 \end{pmatrix}$$

$$T = T_{N-1} T_{N-2} \cdots T_0$$

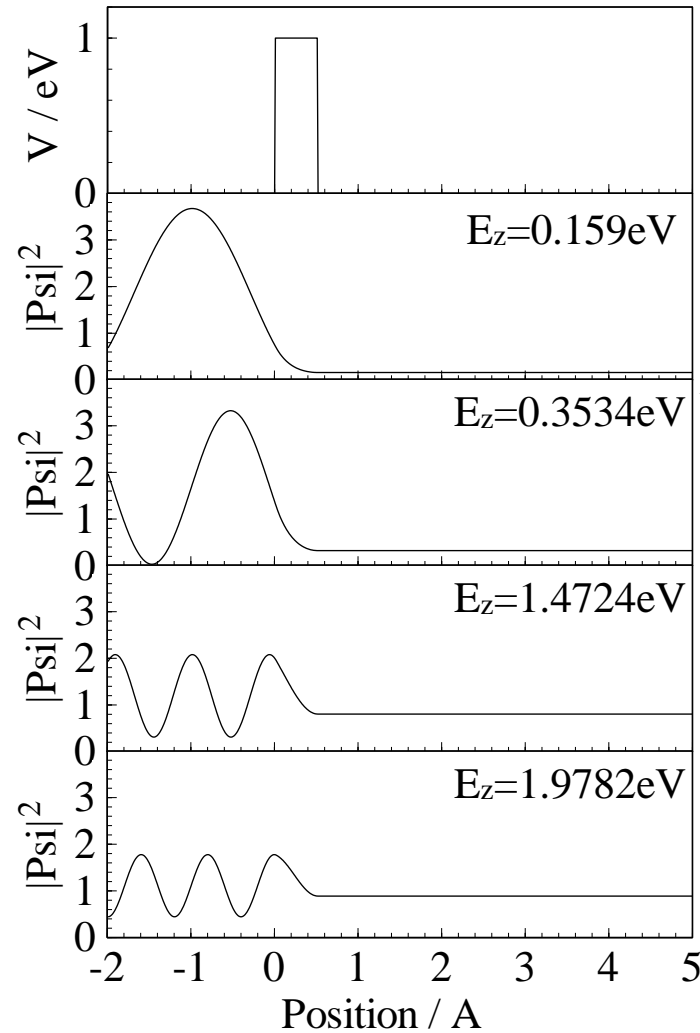
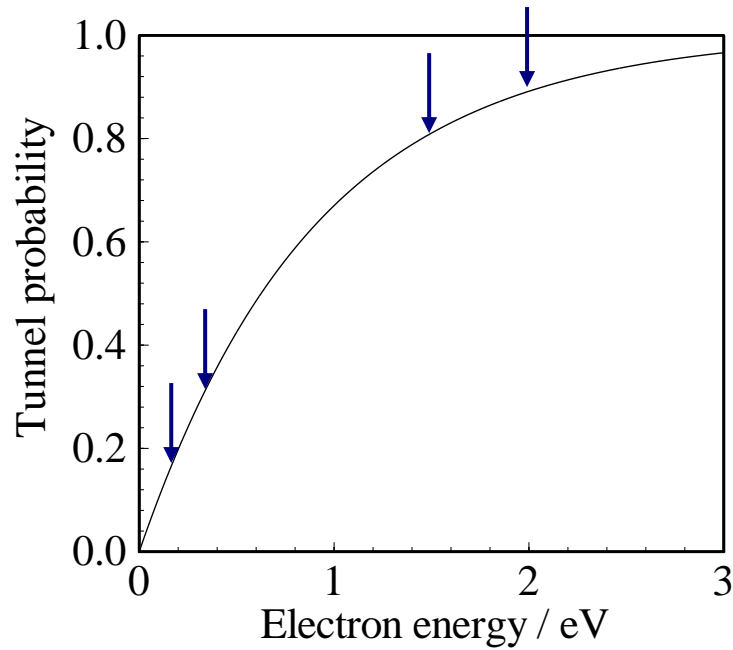
境界条件例:

放出側 ($i = 0$) では
進行波のみが残る

$$A_0 = 1, B_0 = 0$$

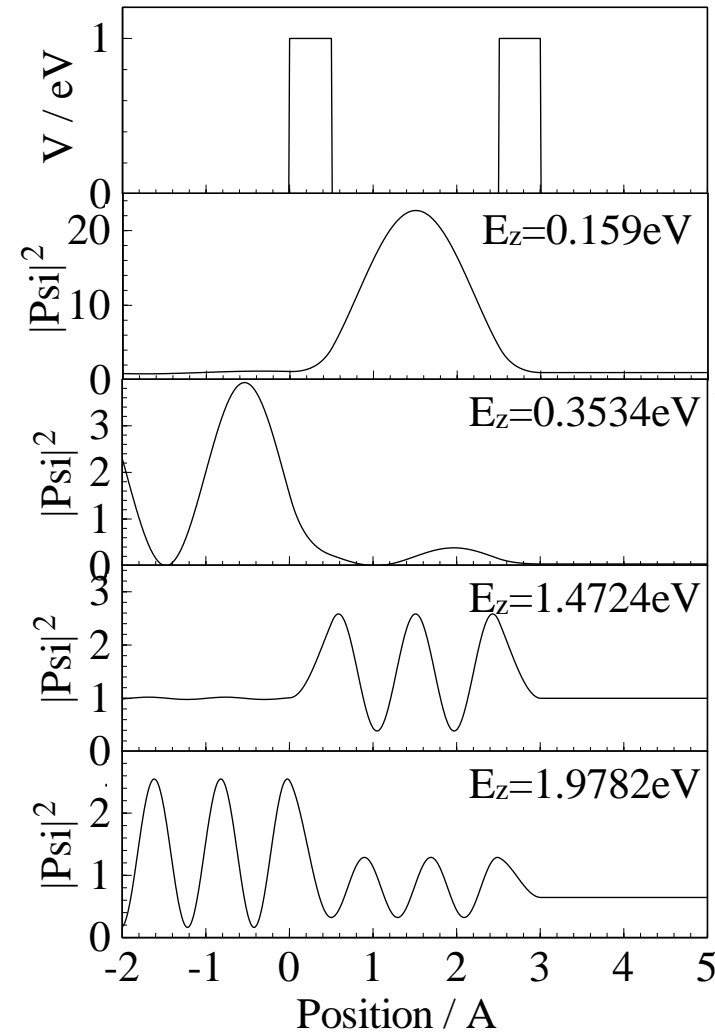
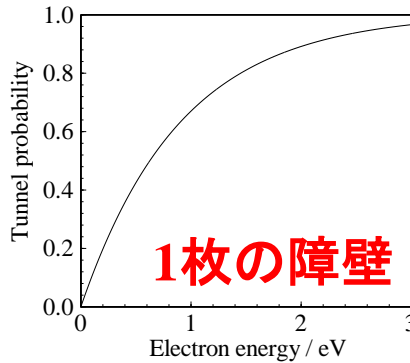
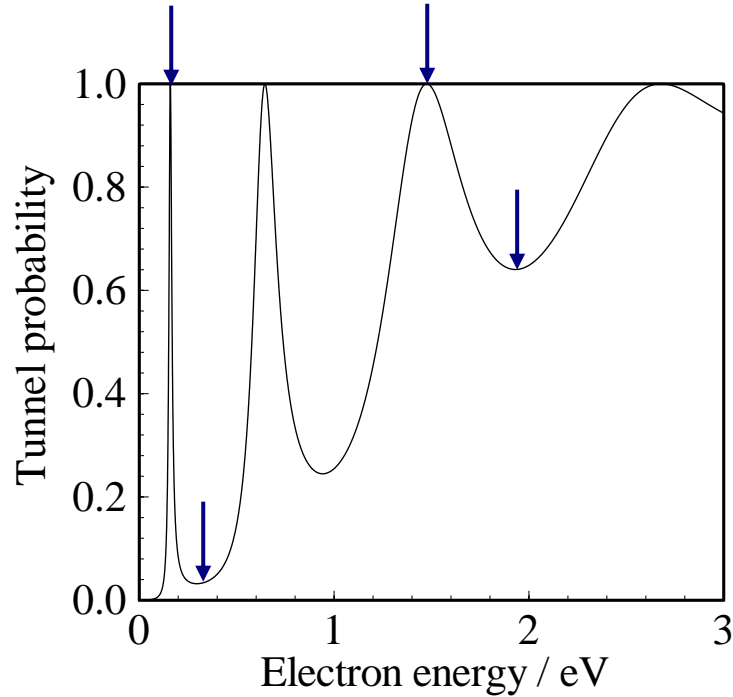


1枚の障壁のトンネル



原子 (障壁) による散乱で、透過率は必ず 1 より小さい
=> 原子がたくさんあったら、透過率は 0 になる？

2枚の障壁のトンネル (QW, RTD)

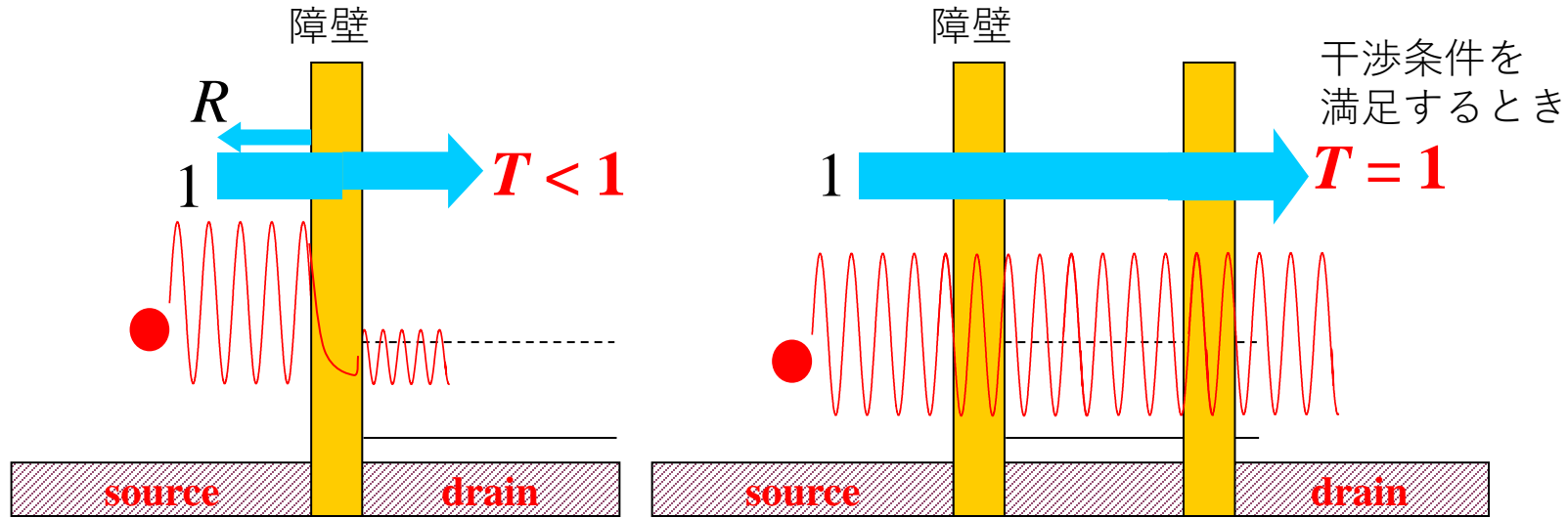


原子がたくさんあったら、透過率は 0 になる？

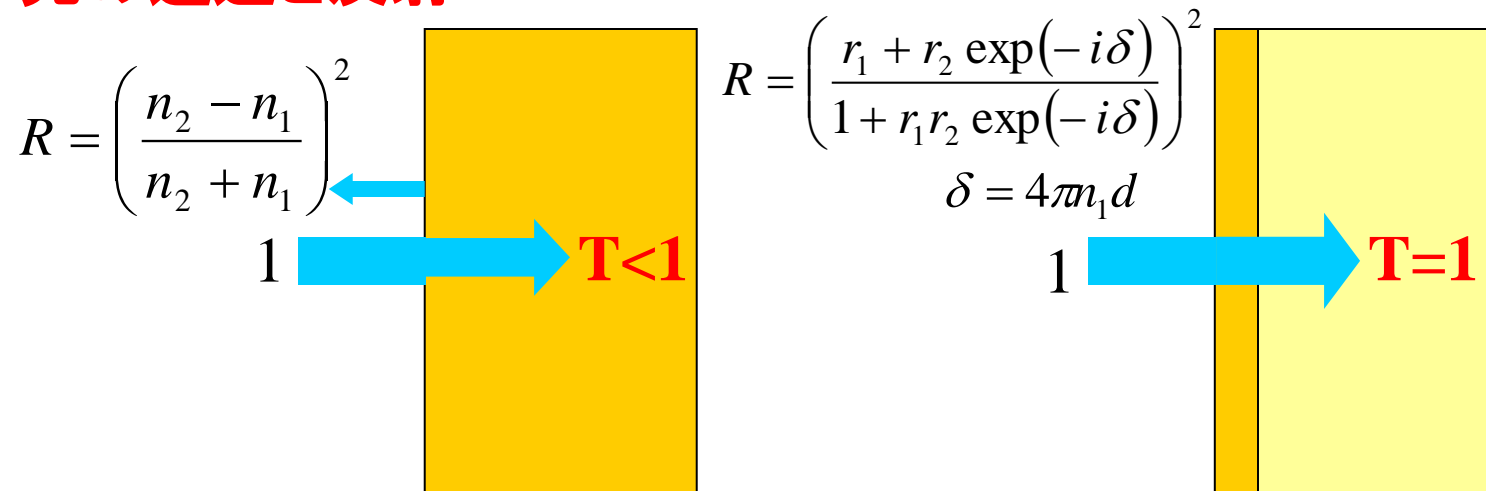
=> 原子 (障壁) が 2つ以上あれば、特定のエネルギーで 100% 透過する

電子と光の散乱

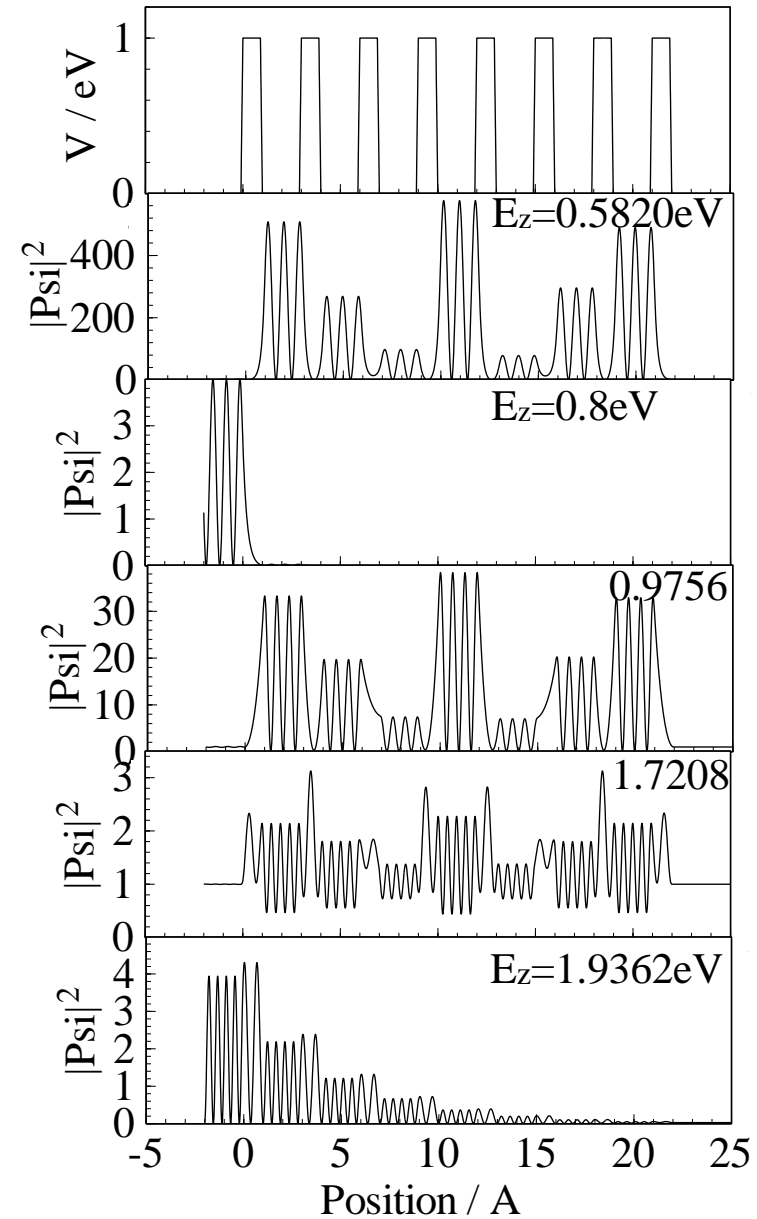
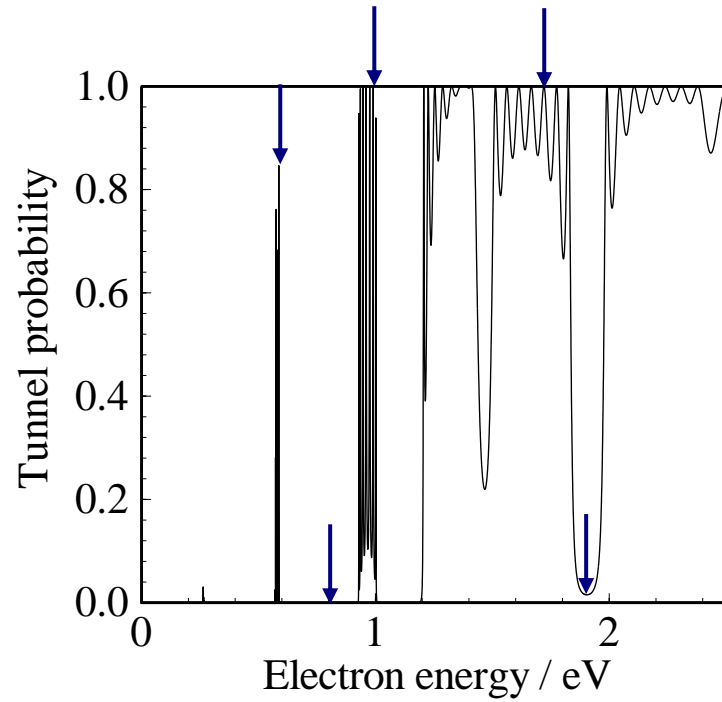
電子の透過と反射



光の透過と反射



多重量子井戸 (MQW) の透過: バンド



Fe/MgO/Fe TMR素子のスピン依存透過率

W.H. Butler, X.-G. Zhang and T.C. Schulthess, Spin-dependent tunneling conductance of Fe|MgO|Fe sandwiches

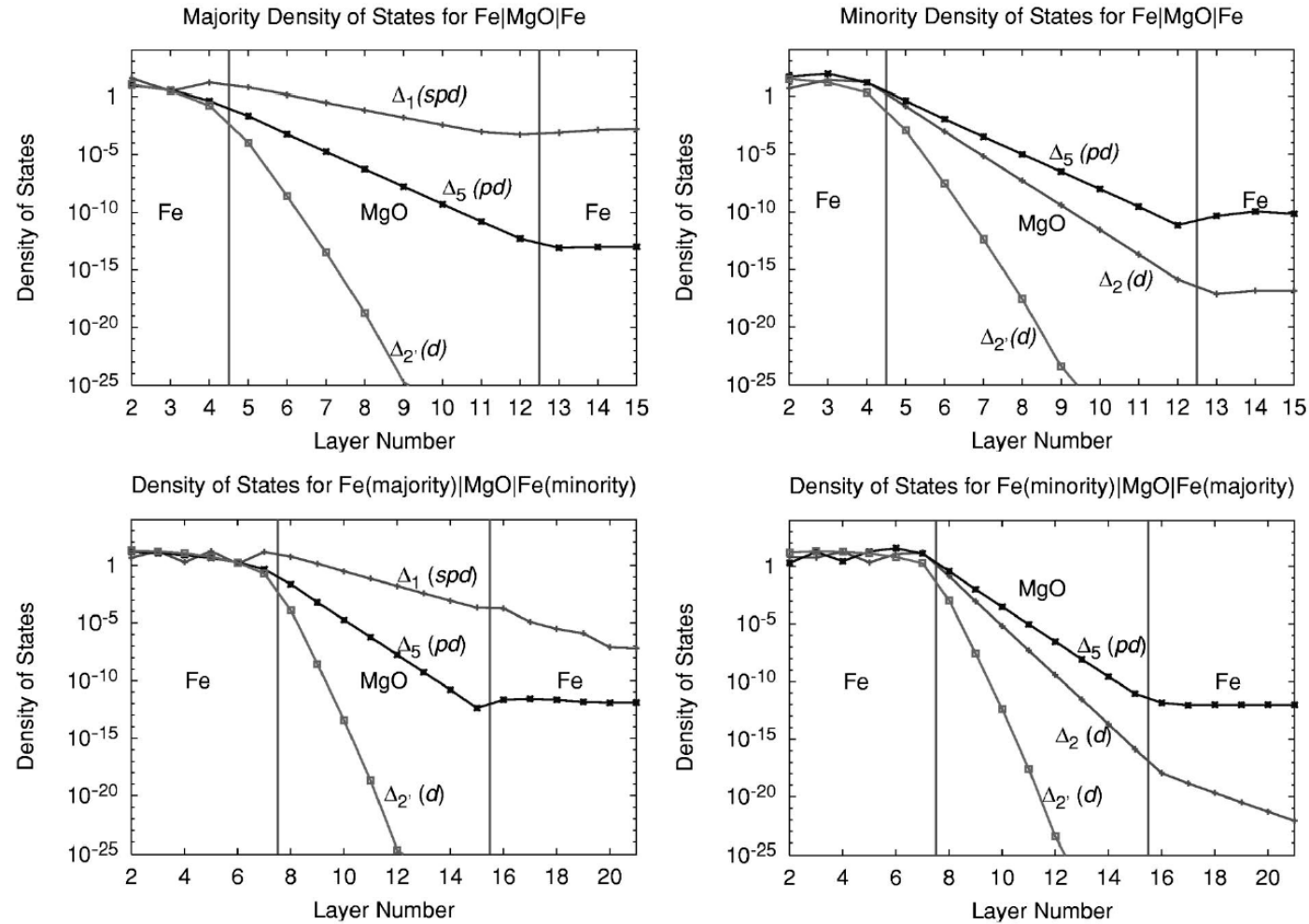
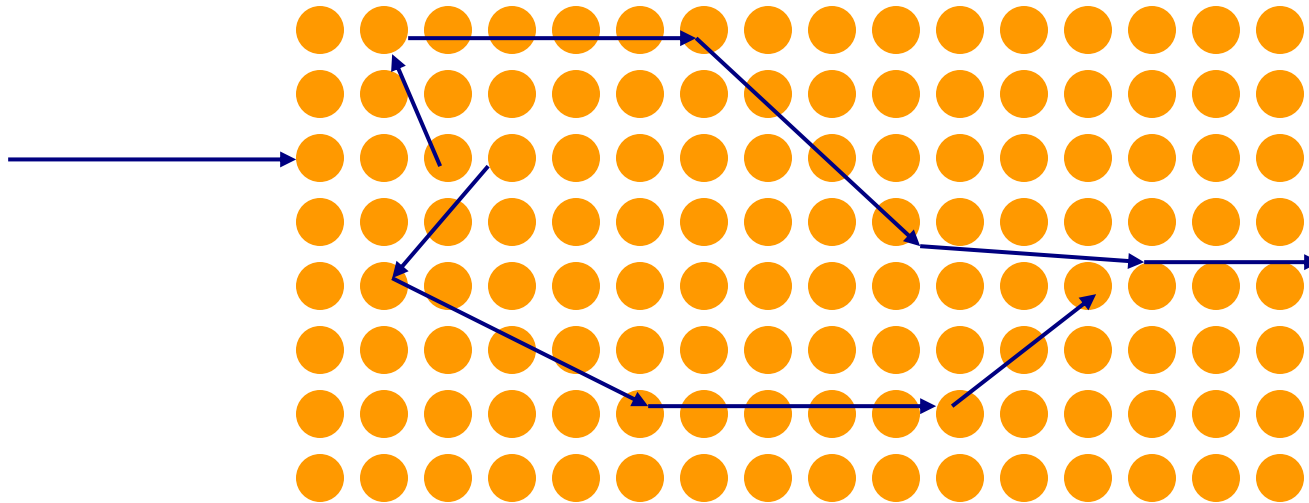


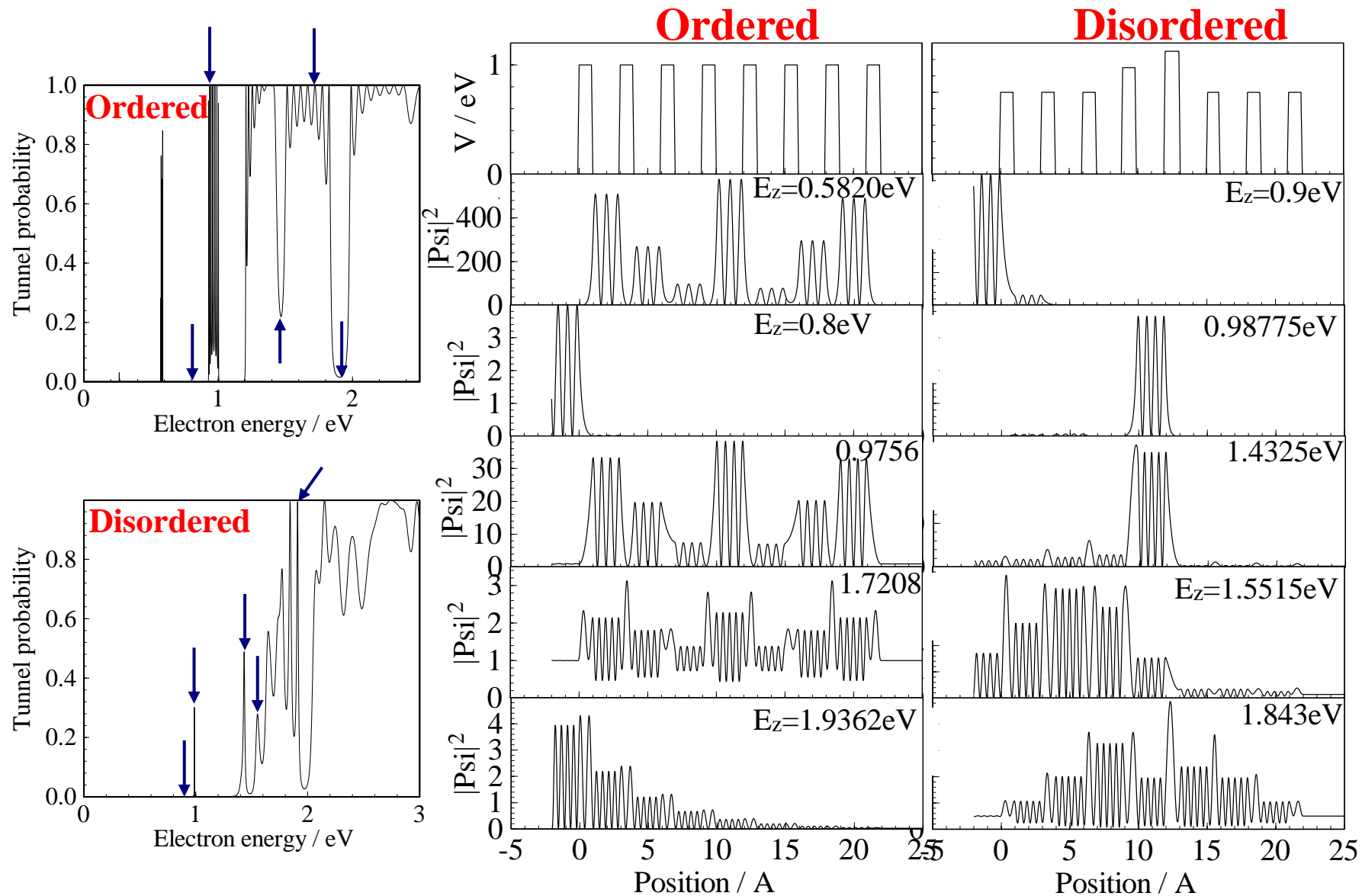
FIG. 7. Tunneling DOS for $k_{\parallel}=0$ for Fe(100)|8MgO|Fe(100). The four panels show the tunneling DOS for majority (upper left) minority (upper right), and antiparallel alignment of the moments in the two electrodes (lower panels). Additional Fe layers are included in the lower panels to show the TDOS variation in the Fe. Each TDOS curve is labeled by the symmetry of the incident Bloch state in the left Fe electrode.

結晶における電子の透過

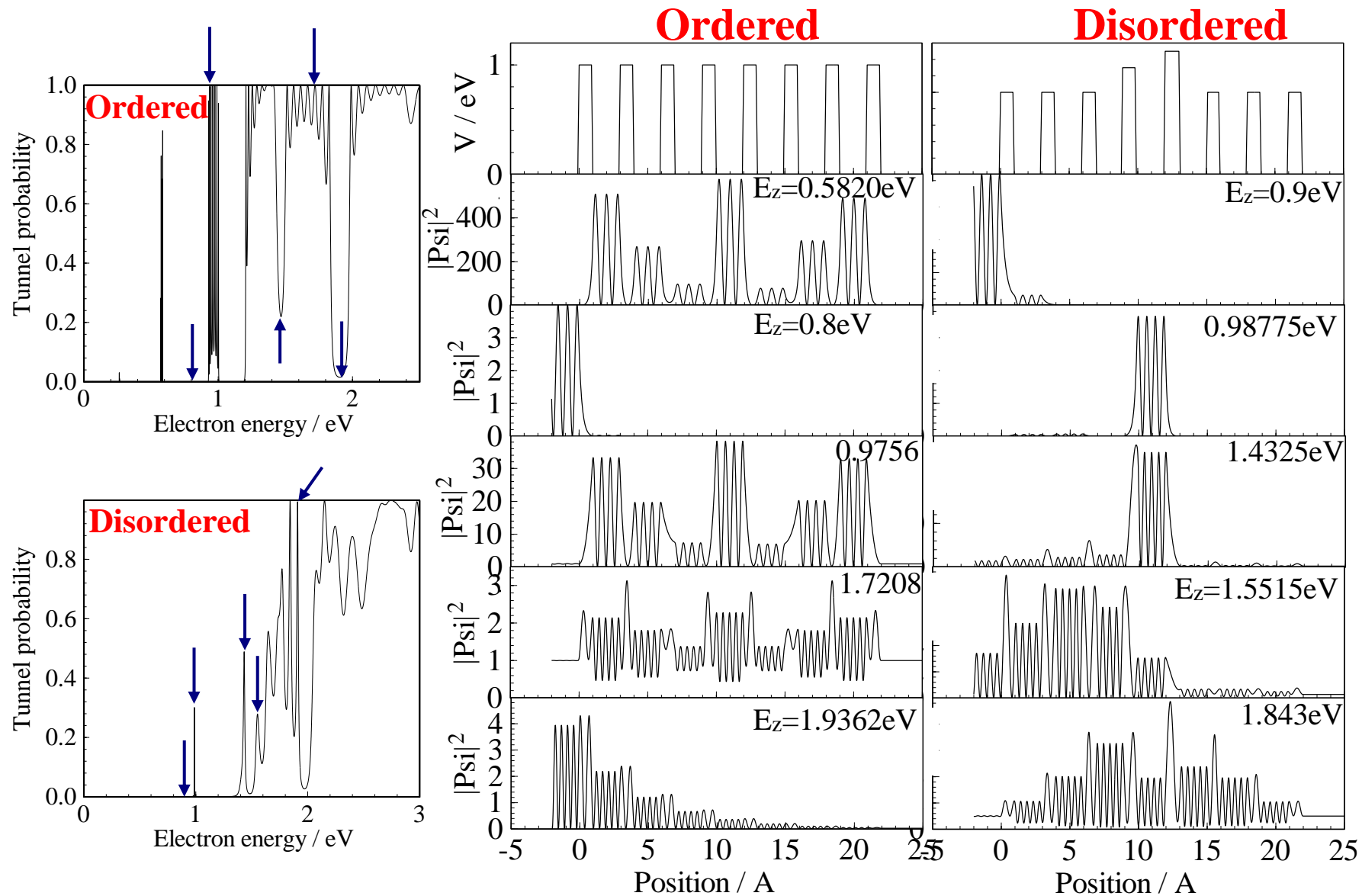
- ・電子が結晶を透過できる ($T = 1$) のは、
三次元に配列した原子からの散乱波が干渉する結果
- ・バンド構造は、透過できる状態のみを表示
- ・任意の運動エネルギーにおいて状態は存在する
ただし、そのほとんどは減衰(散乱)を伴う



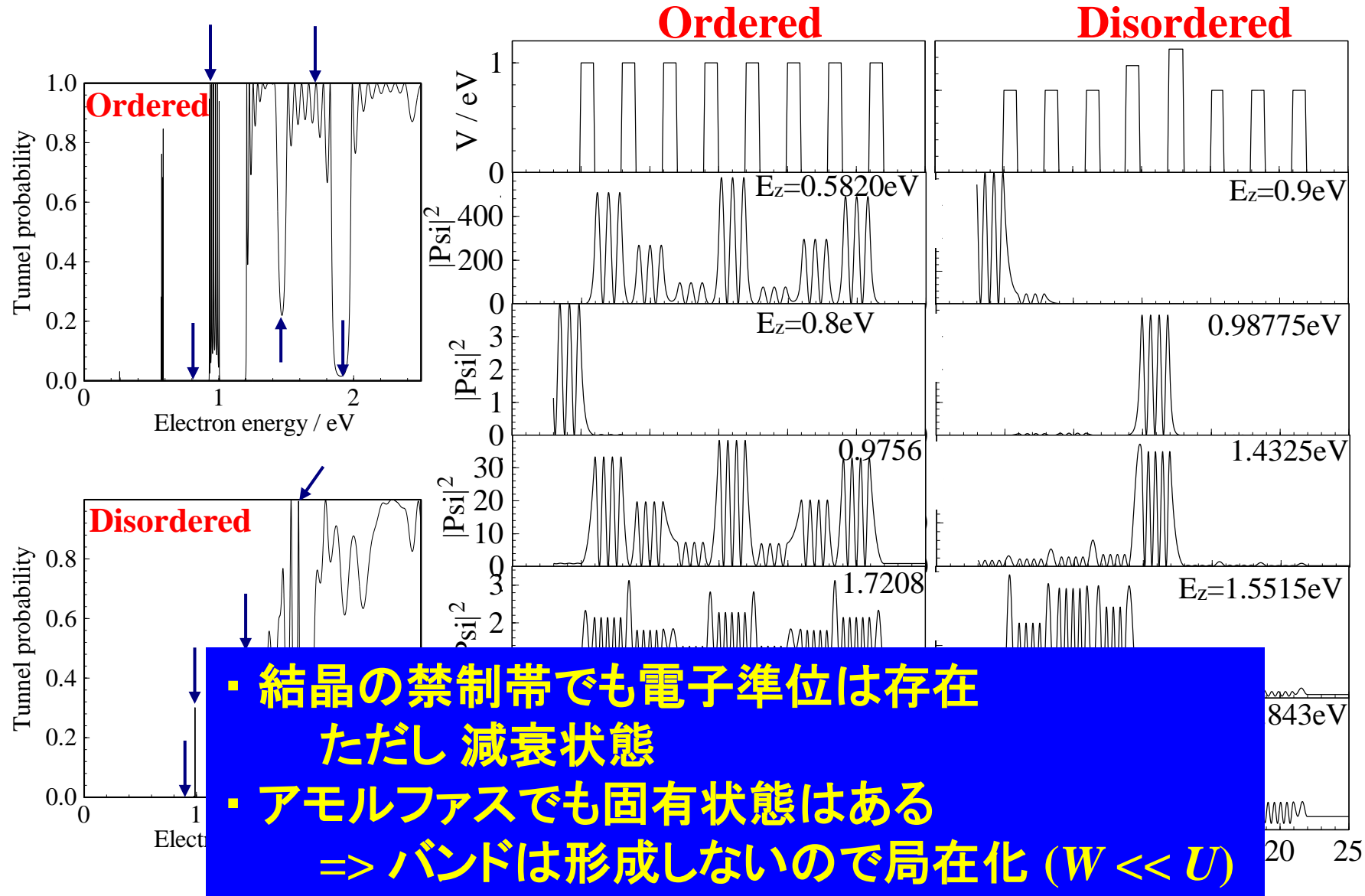
欠陥のある多重量子井戸における電子の透過



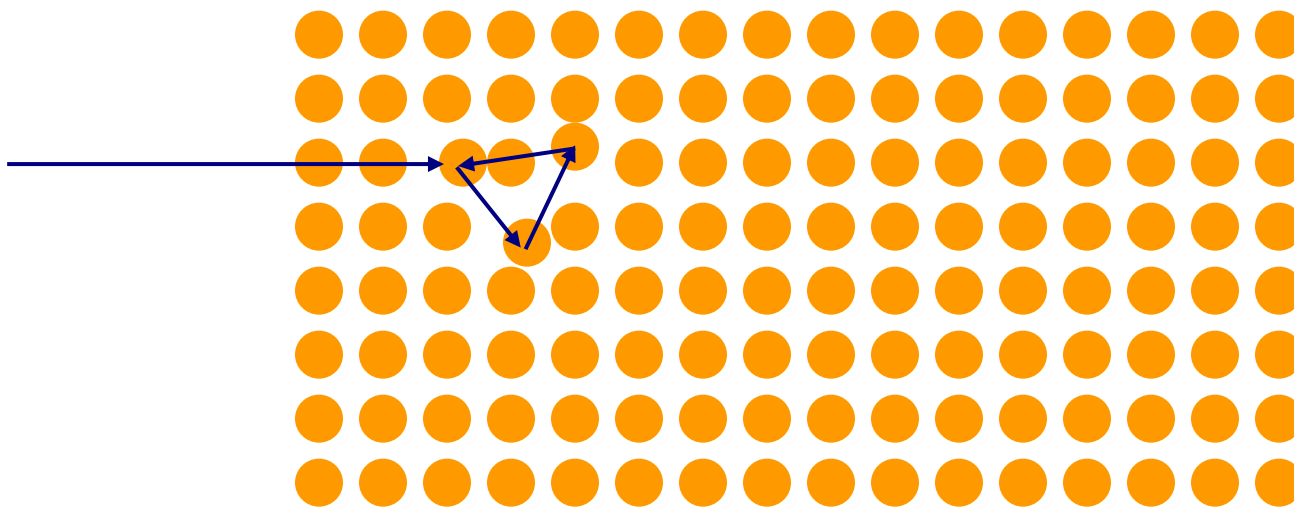
欠陥のある多重量子井戸における電子の透過



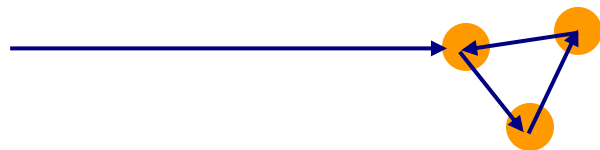
欠陥のある多重量子井戸における電子の透過



乱れのある結晶における電子の透過



- ・背景の結晶部分は電子の透過だけに寄与するので差分だけ考える



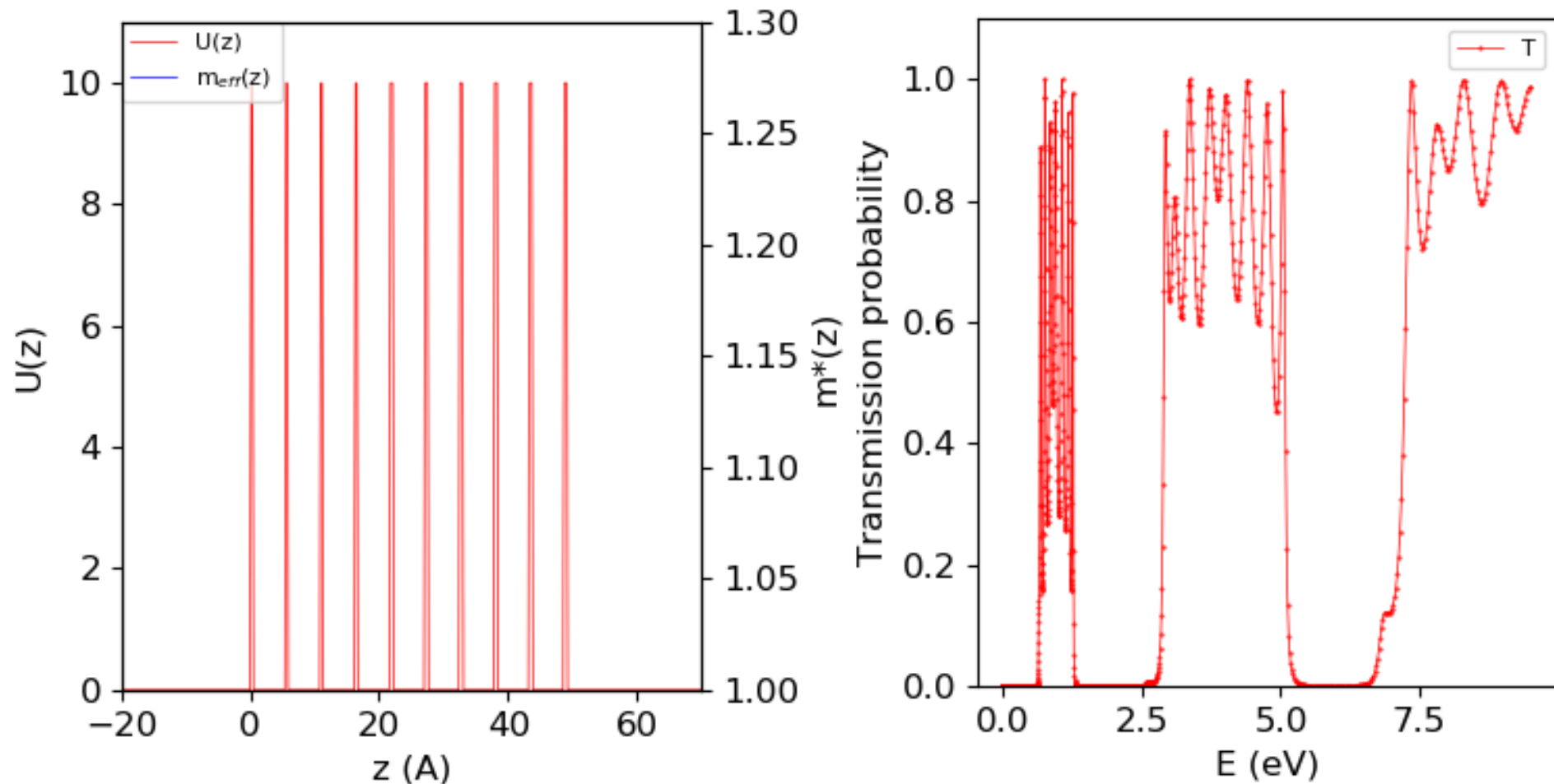
- ・乱れた構造による散乱と干渉の結果、定在波をつくる
アンダーソン局在

プログラム: 転送行列法

Transfer_matrix.py

Si の格子定数 $a = 5.4064 \text{ \AA}$ $m^* = 1.0m_e$
障壁幅 0.5 \AA 障壁高さ 10.0 eV 10周期

`python transfer_matrix.py tr 501 0.1 0.01 9.5 2001`



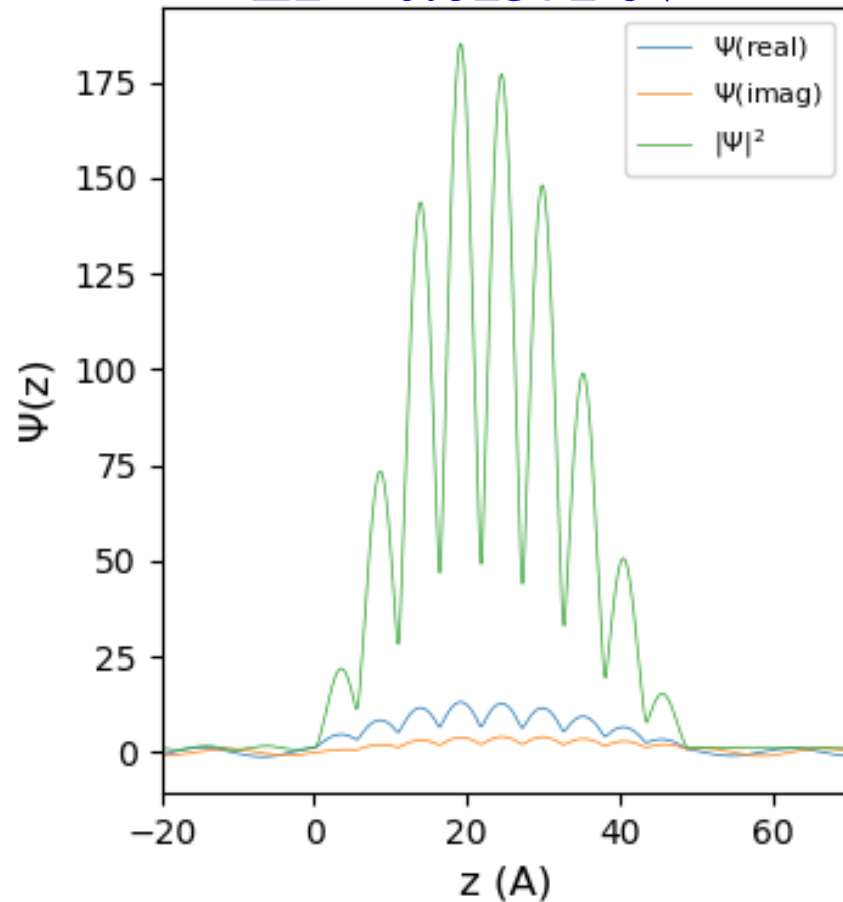
プログラム: 転送行列法

Transfer_matrix.py

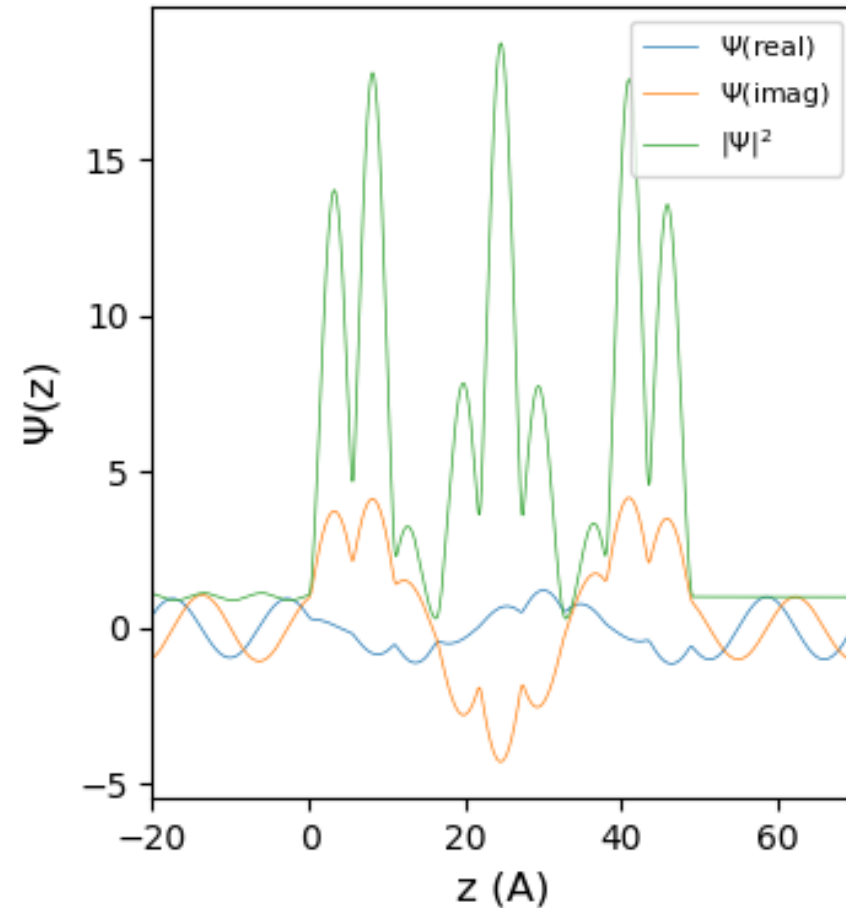
Si の格子定数 $a = 5.4064 \text{ \AA}$ $m^* = 1.0m_e$
障壁幅 0.5 \AA 障壁高さ 10.0 eV 10周期

python transfer_matrix.py wf 5001 Ez

Ez = 0.61371 eV



Ez = 0.701645 eV



常微分方程式の境界値問題: Thomas-Fermiモデル

後藤憲一 他, 詳解現代物理学演習、共立出版 (1972)

$\phi(r)$: 遮蔽された原子核ポテンシャル

$$\phi(r) \rightarrow \frac{1}{4\pi\epsilon_0} \frac{Ze}{r} \quad (r \rightarrow 0)$$
$$0 \quad (r \rightarrow \infty)$$

規格化

$$\chi(r) = \frac{4\pi\epsilon_0}{Ze} r \phi(r) = \frac{4\pi\epsilon_0}{Ze} r (E_F / e + \phi(r))$$

$$r = by = 0.8853 Z^{-1/3} a_0 y$$

$$b = Z^{-1/3} \left(\frac{3\pi}{4} \right)^{2/3} \frac{a_0}{2}$$

$$a_0 = \frac{4\pi\epsilon_0 \hbar^2}{me^2}$$
$$= 0.52921 \text{ \AA}$$

電子密度による近似 (Thomas-Fermiモデル)

$$y^{1/2} \frac{d^2 \chi}{dy^2} = \chi^{3/2} \quad \chi(r) \rightarrow 1 \quad (r \rightarrow 0)$$
$$0 \quad (E_F = 0, r \rightarrow \infty)$$

常微分方程式の境界値問題: Thomas-Fermiモデル

後藤憲一 他, 詳解現代物理学演習、共立出版 (1972)

$$\frac{d^2 \chi}{dy^2} = y^{-1/2} \chi^{3/2} \quad \chi_{n+1} - 2\chi_n + \chi_{n-1} = h^2 \chi''_n + O(h^4) = h^2 y_n^{-1/2} \chi_n^{3/2} + O(h^4)$$

$$\chi_{n+1} = 2\chi_n - \chi_{n-1} + h^2 y_n^{-1/2} \chi_n^{3/2}$$

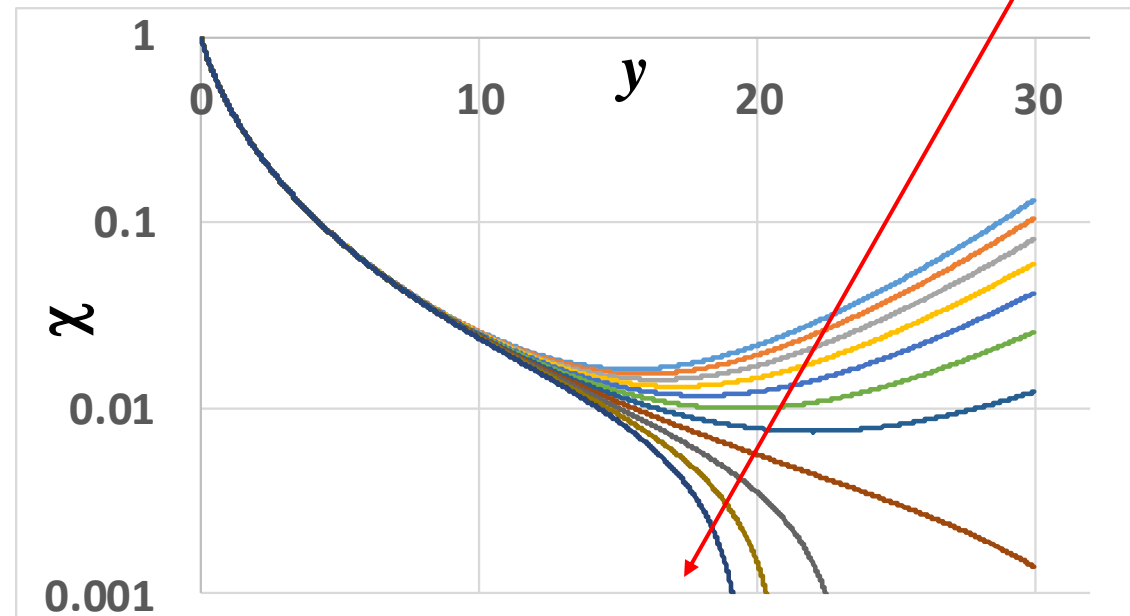
初期条件: $\chi_0 = 1$ $\chi_1 = 1 - \alpha$

境界条件:

$$\chi_n > 0, \quad |\chi_n| < \text{EPS}$$

$$\chi'_n < 0, \quad |\chi'_n| < \text{EPS}'$$

$$\alpha = 0.01442860 \\ \sim 0.01442869$$



常微分方程式の境界値問題: ノイメロフ積分

菅野暁 監修, 足立裕彦、塚田 著, スレーター分子軌道計算, 第3章

東京大学出版会 (1982)

原子のSchrödinger方程式の動径関数 (Rydberg単位)

$$-\frac{1}{r^2} \frac{d}{dr} \left(r^2 \frac{dR(r)}{dr} \right) + \left[-\varepsilon + V(r) + \frac{l(l+1)}{r^2} \right] R(r) = 0 \quad \lim_{r \rightarrow 0, \infty} R(r) = 0$$

$$P(r) = rR(r)$$

$$\frac{d^2}{dr^2} P(r) = g(r)P(r) \quad \begin{aligned} g(r) &= -\varepsilon + V(r) + \frac{l(l+1)}{r^2} \\ &= -\varepsilon - 2\frac{Z}{r} + \frac{l(l+1)}{r^2} \end{aligned}$$

$$P_{n+1} - 2P_n + P_{n-1} = h^2 P''_n + O(h^4) = h^2 g_n P_n + O(h^4) \quad \begin{array}{l} \text{中央の式までは} \\ \text{Verlet法} \end{array}$$

$$P_{n+1} = (2 + h^2 g_n) P_n - P_{n-1}$$

ノイメロフ (Noumerov) 積分:

$$y_n = P_n - \frac{h^2 P''_n}{12} = P_n \left(1 - \frac{h^2 g_n}{12} \right) \quad \text{として次の式を使うと、さらに精度が上がる}$$

$$y_{n+1} = \left(2 + \frac{h^2 g_n}{1 - h^2 g_n / 12} \right) y_n - y_{n-1} + O(h^6)$$

常微分方程式の境界値問題: 波動関数

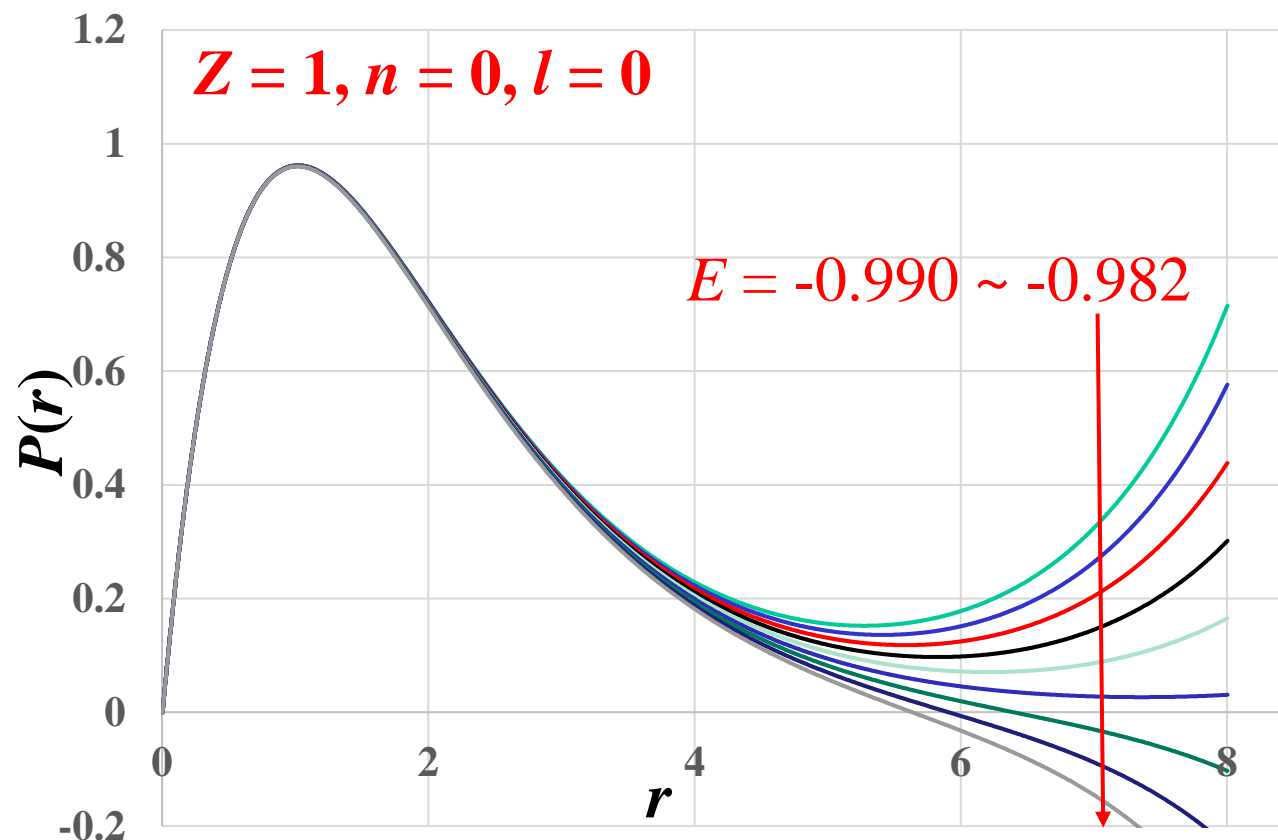
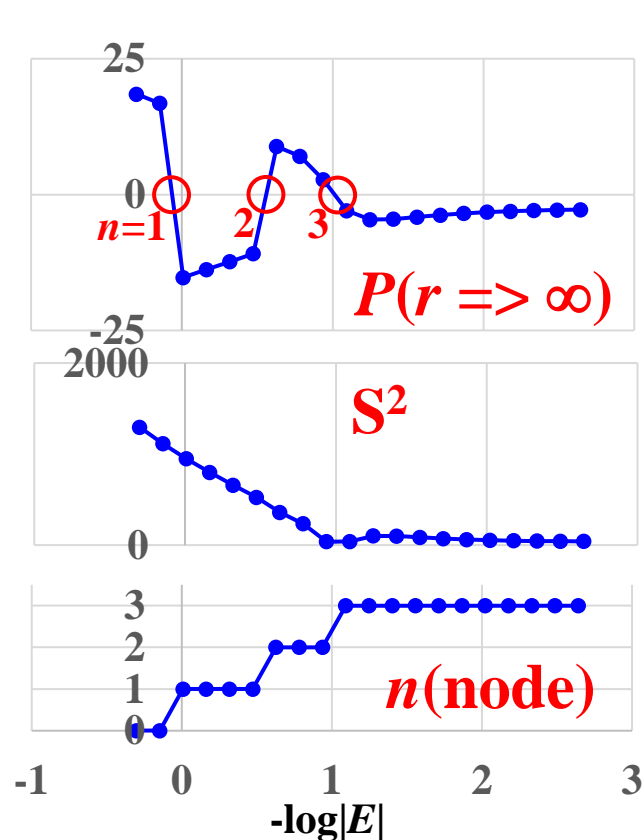
菅野暁 監修, 足立裕彦、塚田 著, スレーター分子軌道計算, 第3章
東京大学出版会 (1982)

$$P(r) = rR(r)$$

$$P_{n+1} = (2 + h^2 g_n) P_n - P_{n-1} \quad g_n = -E - 2 \frac{Z}{r_n} + \frac{l(l+1)}{r_n^2}$$

初期条件: $P_0 = 0, P_1 = \alpha$

境界条件: $\lim_{r \rightarrow 0, \infty} P(r) = 0$



常微分方程式の境界値問題: 波動関数

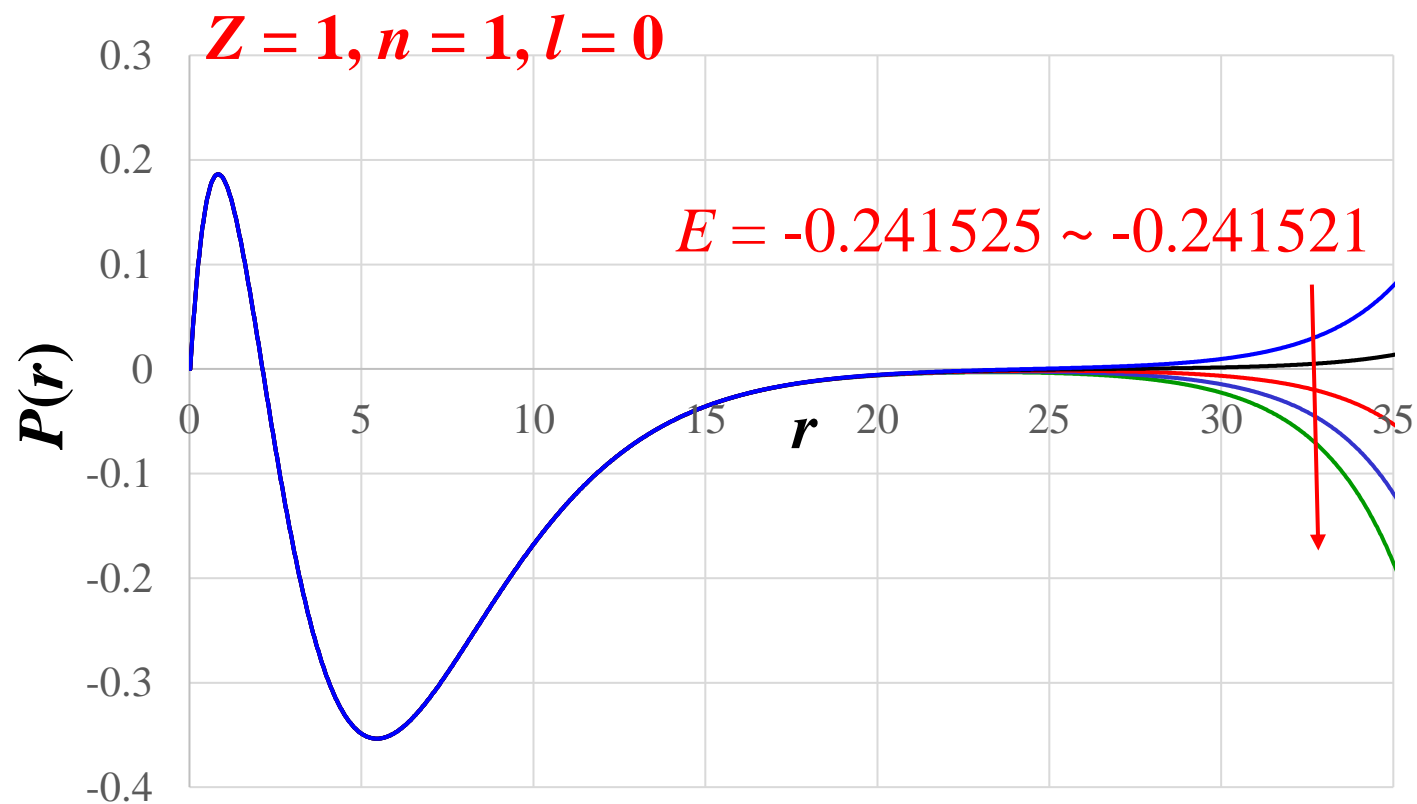
菅野暁 監修, 足立裕彦、塚田 著, スレーター分子軌道計算, 第3章
東京大学出版会 (1982)

$$P(r) = rR(r)$$

$$P_{n+1} = (2 + h^2 g_n)P_n - P_{n-1} \quad g_n = -E - 2\frac{Z}{r_n} + \frac{l(l+1)}{r_n^2}$$

初期条件: $P_0 = 0, P_1 = \alpha$

境界条件: $\lim_{r \rightarrow 0, \infty} P(r) = 0$



Kramers-Kronig Transformation

クラマースークローニツヒ (KK) の関係式

因果律が成立していれば

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

$$\begin{aligned}\varepsilon_r &= 1 + \frac{1}{\pi} P \int_{-\infty}^{\infty} \frac{\omega' \varepsilon_i(\omega')}{\omega'^2 - \omega^2} d\omega' \left(= 1 + \frac{2}{\pi} P \int_0^{\infty} \frac{\omega' \varepsilon_i(\omega')}{\omega'^2 - \omega^2} d\omega' \right) \\ \varepsilon_i &= -\frac{1}{\pi} P \int_{-\infty}^{\infty} \frac{\varepsilon_r(\omega') - 1}{\omega'^2 - \omega^2} d\omega' \left(= -\frac{2}{\pi} P \int_0^{\infty} \frac{\varepsilon_r(\omega') - 1}{\omega'^2 - \omega^2} d\omega' \right)\end{aligned}$$

インパルス応答が実数の場合、カッコ内の式が使える

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

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

クラマースークローニツヒの関係式

Kramers-Kronig relation

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

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

P : Principal value of the integral

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

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

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

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

にKK変換を当てはめる

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

Optical spectrum (誘電関数 ϵ^* , 吸収係数 α)

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

$$\epsilon_1(\omega) = 1 + 4\pi \sum_j \frac{e^2 |T_{0j}|^2}{\hbar} \frac{2\omega_j}{\omega_j^2 - \omega^2}$$

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

Kramers-Kronig transformation

$$\begin{aligned} \epsilon_2(\omega) &= \frac{4\pi N e^2}{m} \sum_j f_j \pi \delta(\omega^2 - \omega_j^2) \\ &= \frac{4\pi N e^2}{m} \sum_j f_j \frac{\pi}{2\omega} [\delta(\omega - \omega_j) + \delta(\omega + \omega_j)] \end{aligned}$$

$$n(\omega) - i\kappa(\omega) = \sqrt{\epsilon_1(\omega) - i\epsilon_2(\omega)}$$

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

KK transformation: Numerical approach

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

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

**Numerical integration
for given data**

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

Extrapolation to high f

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

Extrapolation to low f

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

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

光学スペクトルのKK変換: 外挿問題

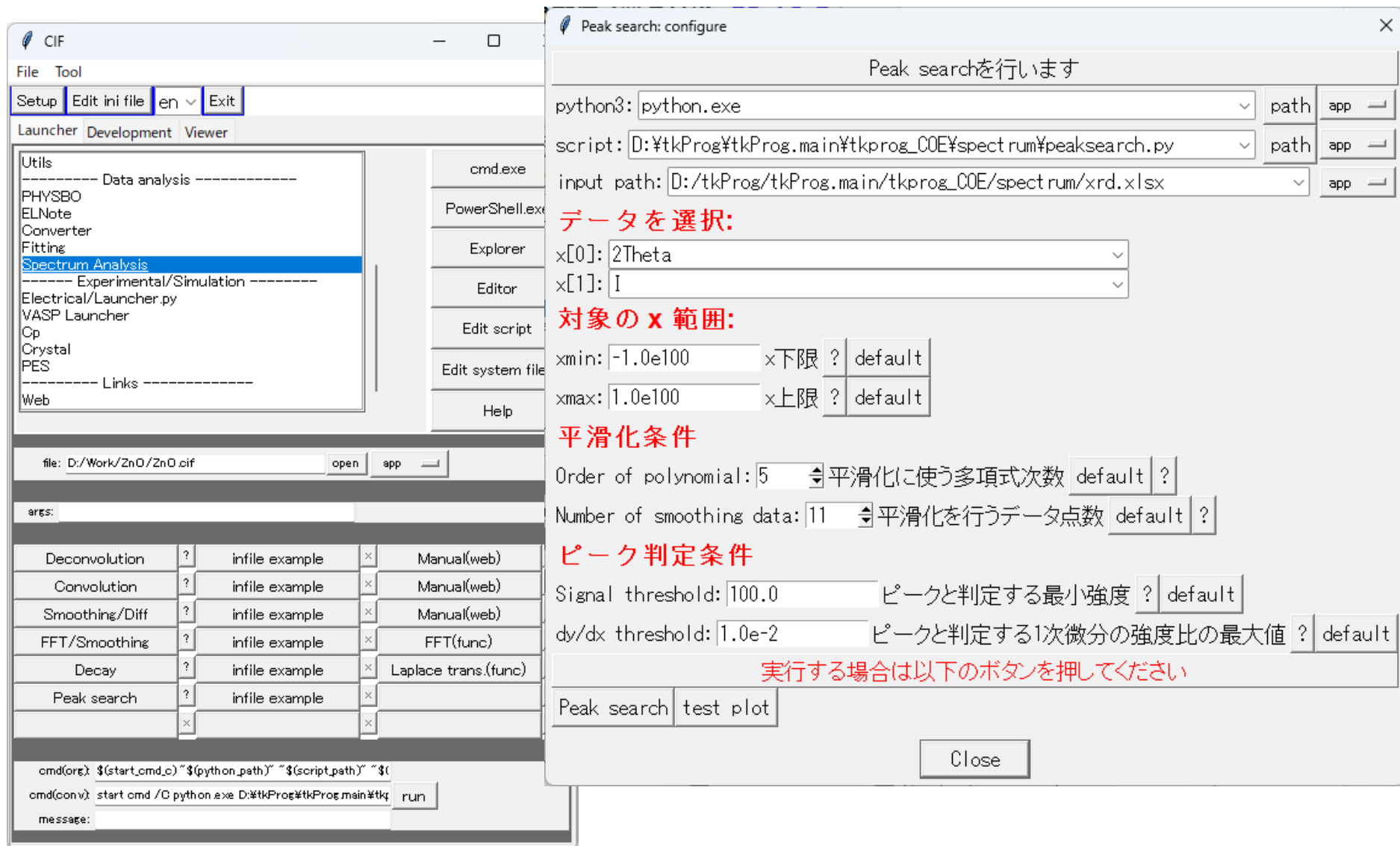
Fourier変換と同様に、測定周波数外の情報がないことが問題
測定周波数外のデータは結果に大きく影響する

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

- ・高エネルギー領域: ν^{-4} で外挿するのが一般的
- ・知りたい領域が0.1eV以下:
 - ・金属の低エネルギー領域: Drude反射率で外挿
 - ・半導体・誘電体の低エネルギー領域:
静的誘電率 ϵ_0 から求めた反射率 で外挿 $(\sqrt{\epsilon_0} - 1)^2 / (\sqrt{\epsilon_0} + 1)^2$
- ・知りたい領域が1eV程度まで:
フォノン分散の始まる0.1eV以下は考慮する必要はない
- ・半導体・誘電体の低エネルギー領域:
高周波誘電率 ϵ_∞ から求めた反射率 で外挿 $(\sqrt{\epsilon_\infty} - 1)^2 / (\sqrt{\epsilon_\infty} + 1)^2$

Q: Peak search program

- <http://conf.msl.titech.ac.jp/D2MatE/PeakSearch/PeakSearch.html>

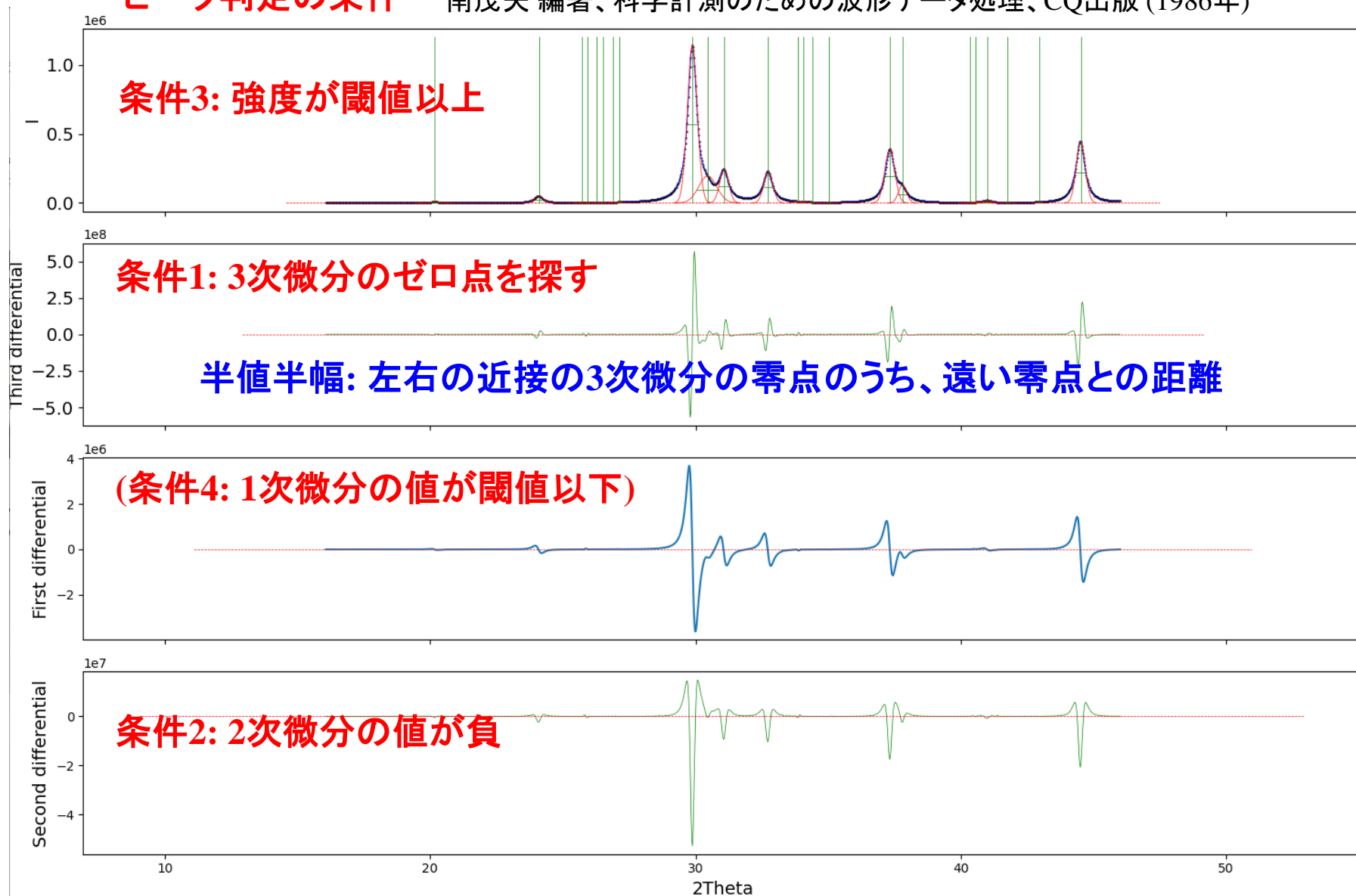


A: Peak search program

[tkProg]¥tkprog_base¥spectrum¥peak_search.py

ピーク判定の条件

南茂夫 編著、科学計測のための波形データ処理、CQ出版 (1986年)



Q: Methfessel-Paxton and tetrahedron method

These are used to integrate / smear $E(k)$ obtained by band calculations

purposes of smearing used in band calculations.

- 1. Increasing the accuracy of 1st BZ integration
(interpolation)**

Tetrahedron method

- 2. Stabilize convergence of SCF (distribution)**

Gauss smearing, Fermi smearing

- 3. Make the DOS display easier to read (smoothing)**

Polynomial fitting may help

**but convolution (smearing) and tetrahedron
method are commonly used**

Q: Determination of E_V (HOMO), E_C (LUMO), E_g

1. Calculate $E(k)$ by band calculation **throughout the first Brillouin zone.**
2. **Sort $E(k)$** from the lowest to the highest
3. Find HOMO (or E_F) level from the number of electrons (N_{tot}) and the number of orbitals included in the calculation
4. Find LUMO level as the next upper $E(k)$ from HOMO

If you have Density-Of-States data $D(E)$,

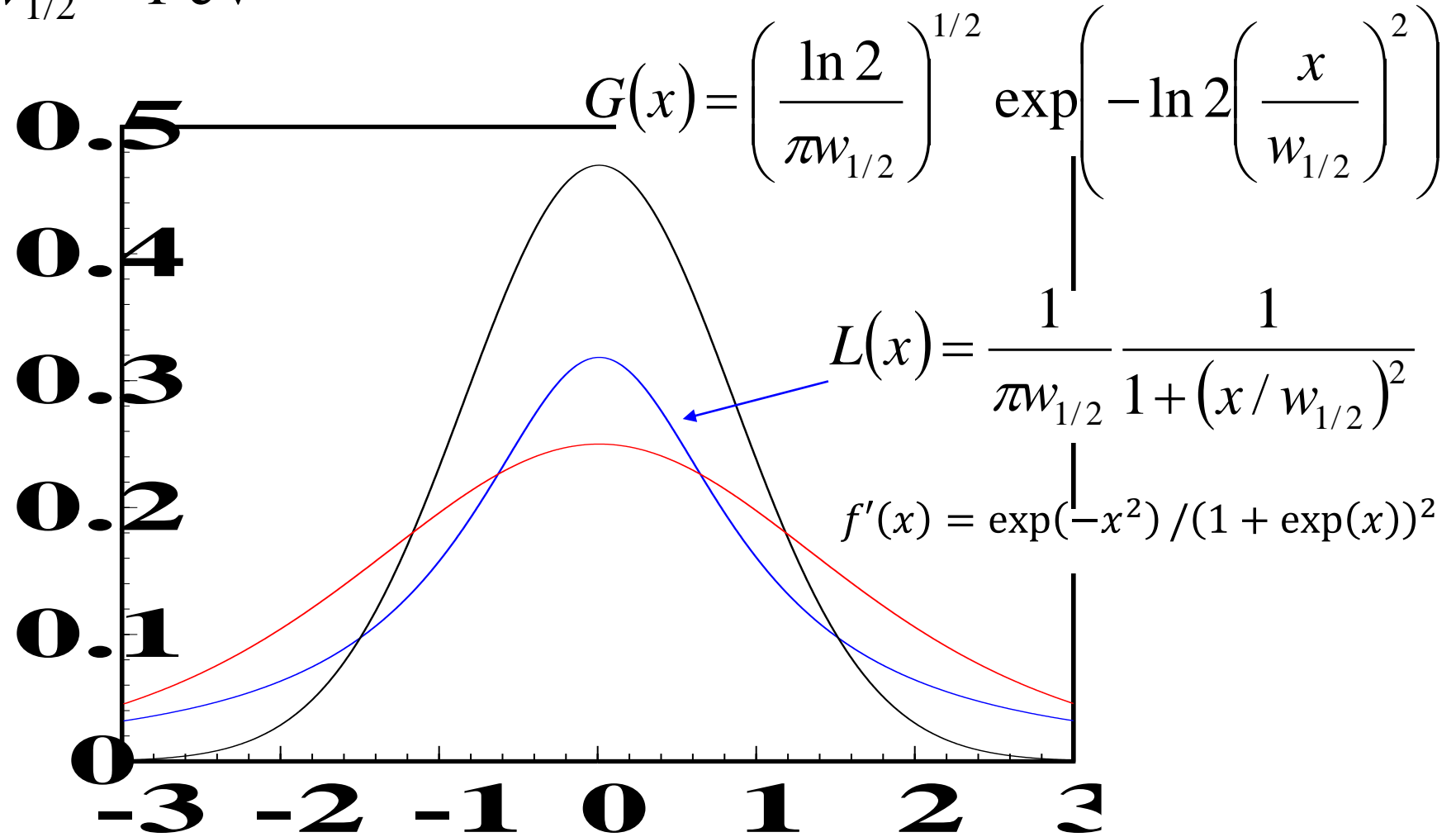
1. Integrate $D(E)$ from the lowest energy to get an integrated electron number function $N(E)$
2. Find HOMO (E_F) as the energy satisfying $N(E_F) = N_{\text{tot}}$
3. Find LUMO level as the next upper $E(k)$ from HOMO

NOTE: If $D(E)$ is smeared, it is difficult to find exact HOMO and LUMO.
Use non-smeared $D(E)$ or tetrahedron-method

For VASP, see [tkProg_Root]¥tkprog_base¥VASP¥gbandedges

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

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

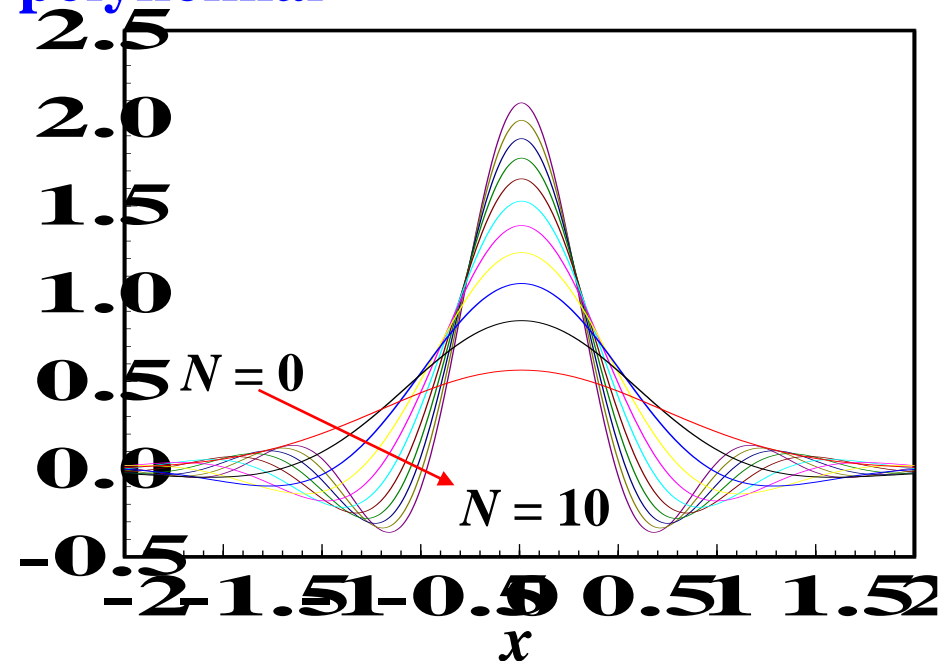
$$D_N(x) = \sum_{n=0}^N A_n H_{2n}(x) \exp(-x^2)$$

**$D_N(x)$ 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

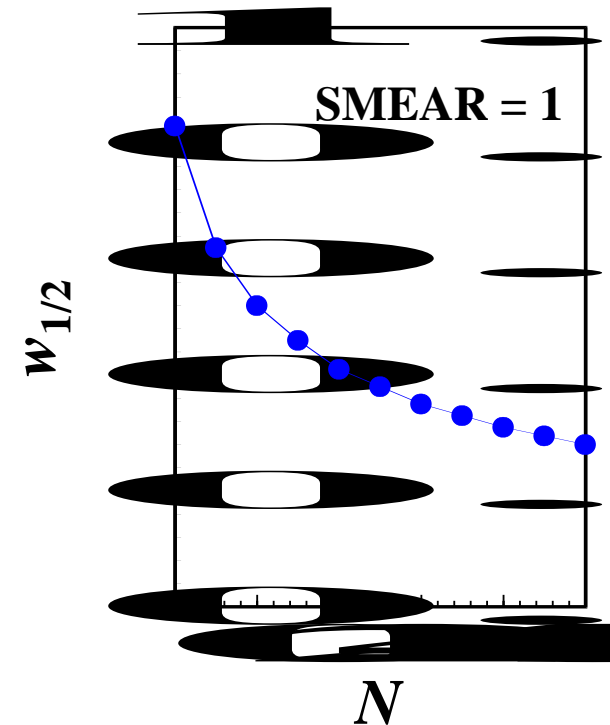
$$\int_{-\infty}^{\infty} H_n(x) H_m(x) \exp(-x^2) dx = \delta_{mn} 2^n \sqrt{\pi} n!$$

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

