

Computational Materials Science (計算材料学特論)

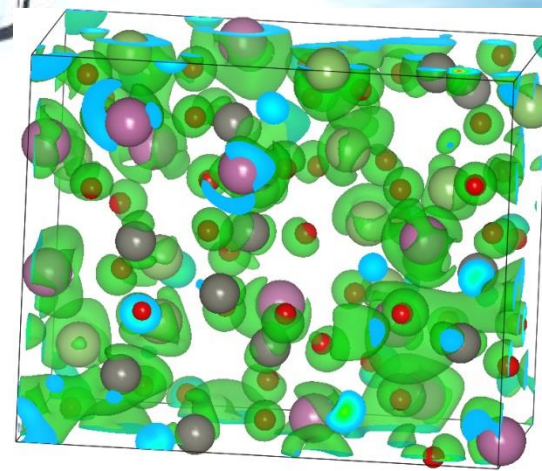
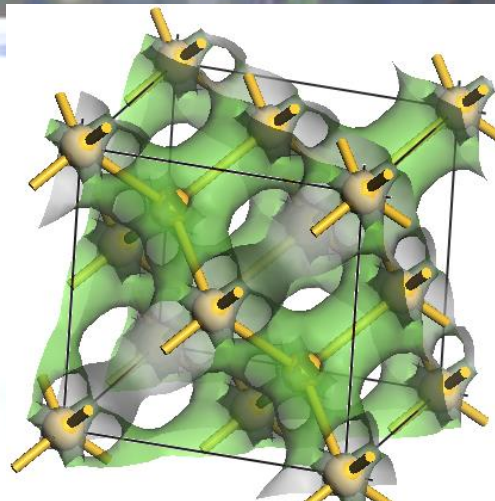
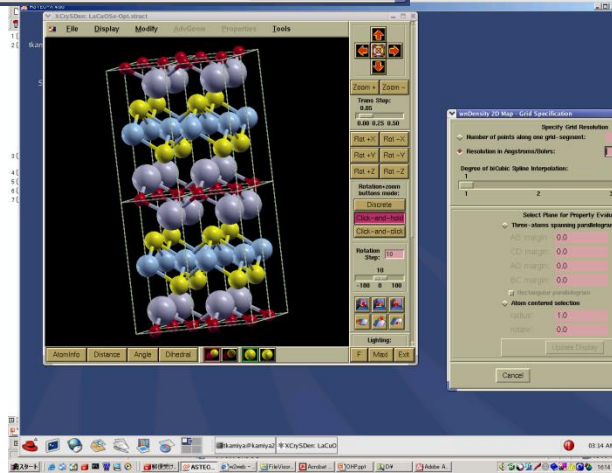
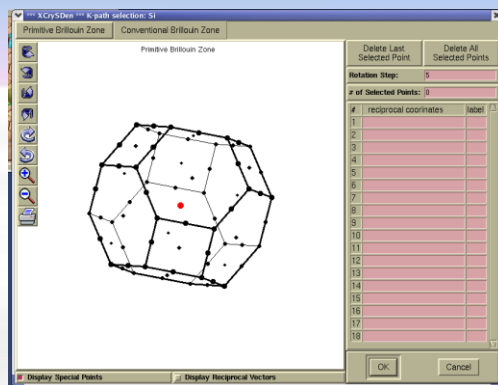
No update today: Lecture materials

Will start ~8:52

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 (Fourier transformation (フーリエ変換), Matrix (行列))
- July 2 (Tue) No lecture (休講)**
- #07 July 5 (Fri) Kamiya, Review (復習)
- #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 21

課題解答の解説

PROBLEM, June 21

PROBLEM:

**Smoothen the data DOS(E) in dos.xlsx
by simple average method and polynomial fit method.**

**Add them and plot the raw DOS(E) and the smoothed data in an
Excel file.**

**You can choose smoothing parameters as you like, but explicitly
describe them.**

Submit the excel file.

See dos_smoothing_answer.xlsx

Smoothing

Simple moving average (2m+1 points)

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

Weighted moving average (2m+1 points)

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

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

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

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

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

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

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

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

Weights for various smoothing

Weighted moving average (2m+1 points)

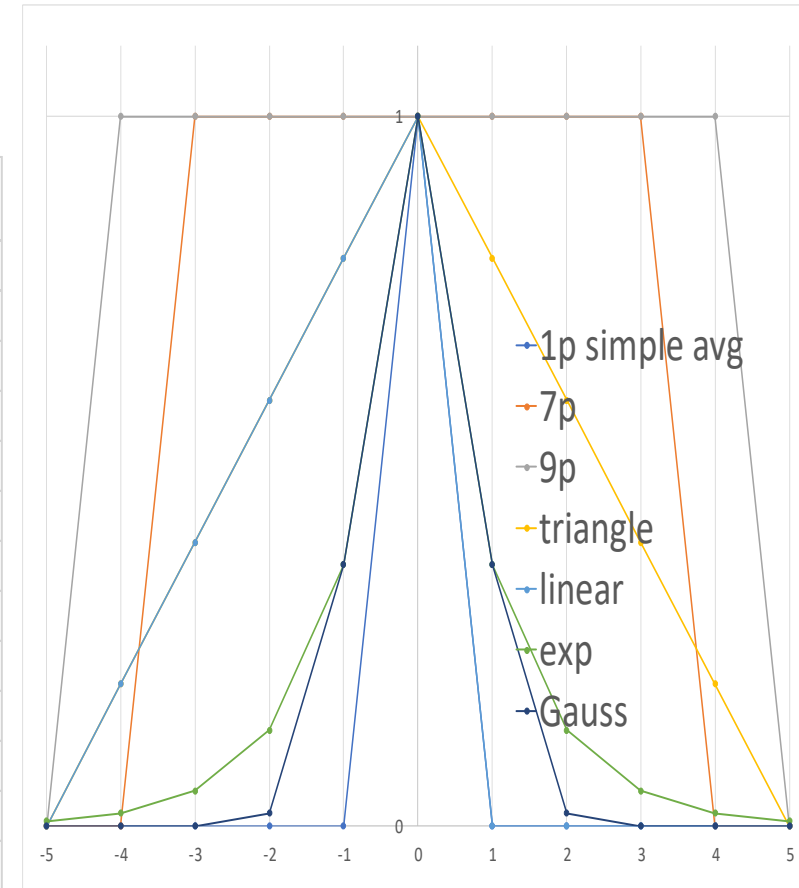
$$y_{i,smoothed} = \frac{1}{\sum_{j=i-m}^{i+m} w_j} \sum_{j=i-m}^{i+m} w_j y_j$$

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

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

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

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

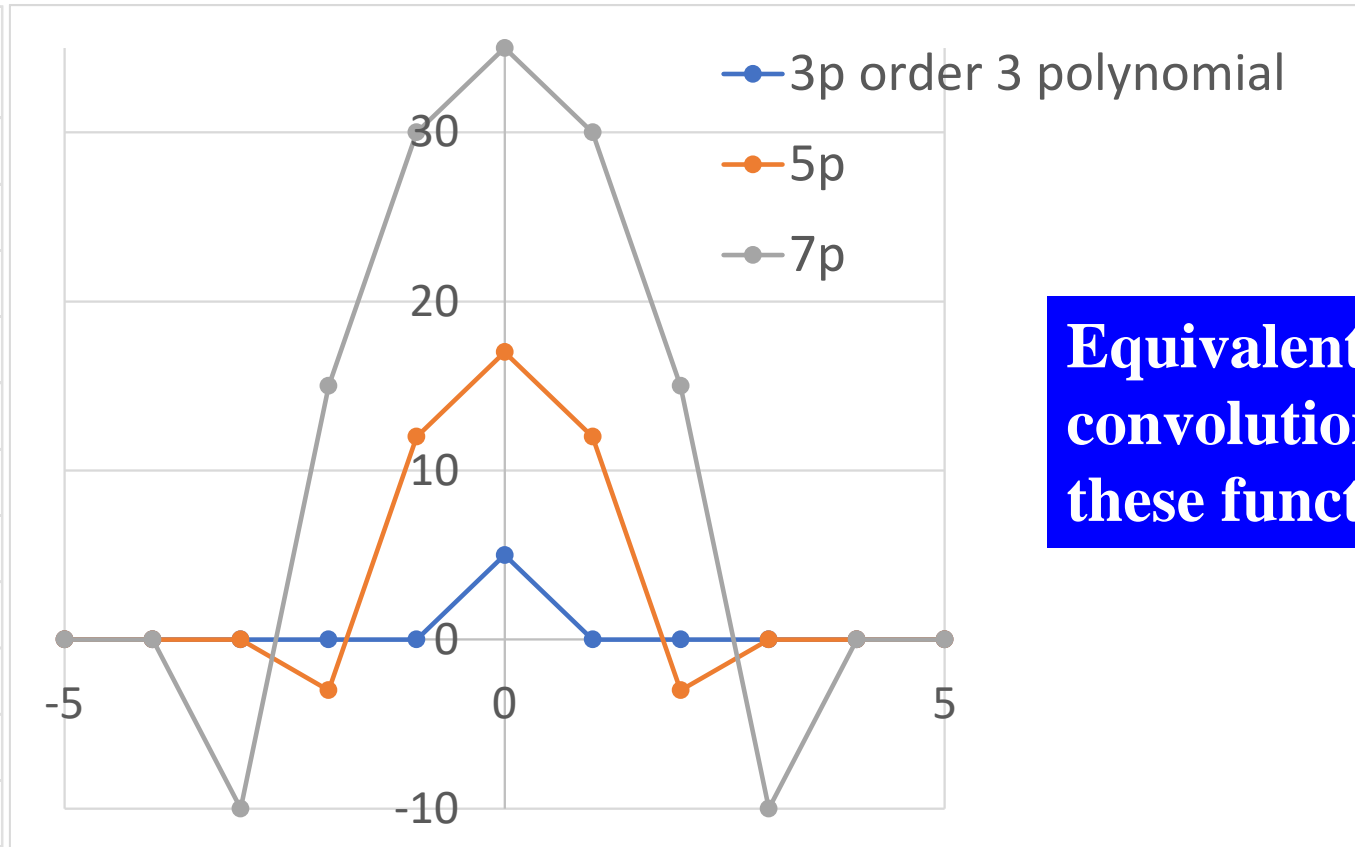


Equivalent to convolution
using these functions

Weights

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

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



Equivalent to convolution using these functions

A smart answer by student

Use Excel matrix functions:

`mmult(range1, range2):`

Multiply matrixes (vectors) given in the ranges range1 and range2

`transpose(range1)`

Transpose the matrix (vector) given in the range1

See “`7p(transpose+mmult)`” column in dos_smoothing_answer.xlsx

PROBLEM, June 25

- 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:

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.
- 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

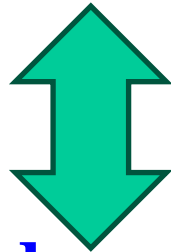
How to solve equations?

Self-consistent method

自己無撞着法

Final unique solution is obtained just by one step calculation

- **Linear least-squares method**
- **Up to 4th order polynomial eq.**



**Five or higher order polynomial,
Transcendental equation (超越方程式)**

- **Difficult to have an analytical solution**
- **Even numerical analysis cannot give final solution by one-cycle calculation**
=> Iterative calculation (反復計算)

Simplest method: Self-consistent (SC) method

A simple case: Solve $g(x) = 0$

SC method is applicable by converting to $x = g(x) + x = f(x)$

Note: not efficient nor stable for many cases

Simple procedure:

Initial value x_0

1st iteration : $x_1 = f(x_0)$

2nd iteration: $x_2 = f(x_1) \dots$

Difficult to converge: Diverge, Oscillation

(収束しにくい: 発散、振動)

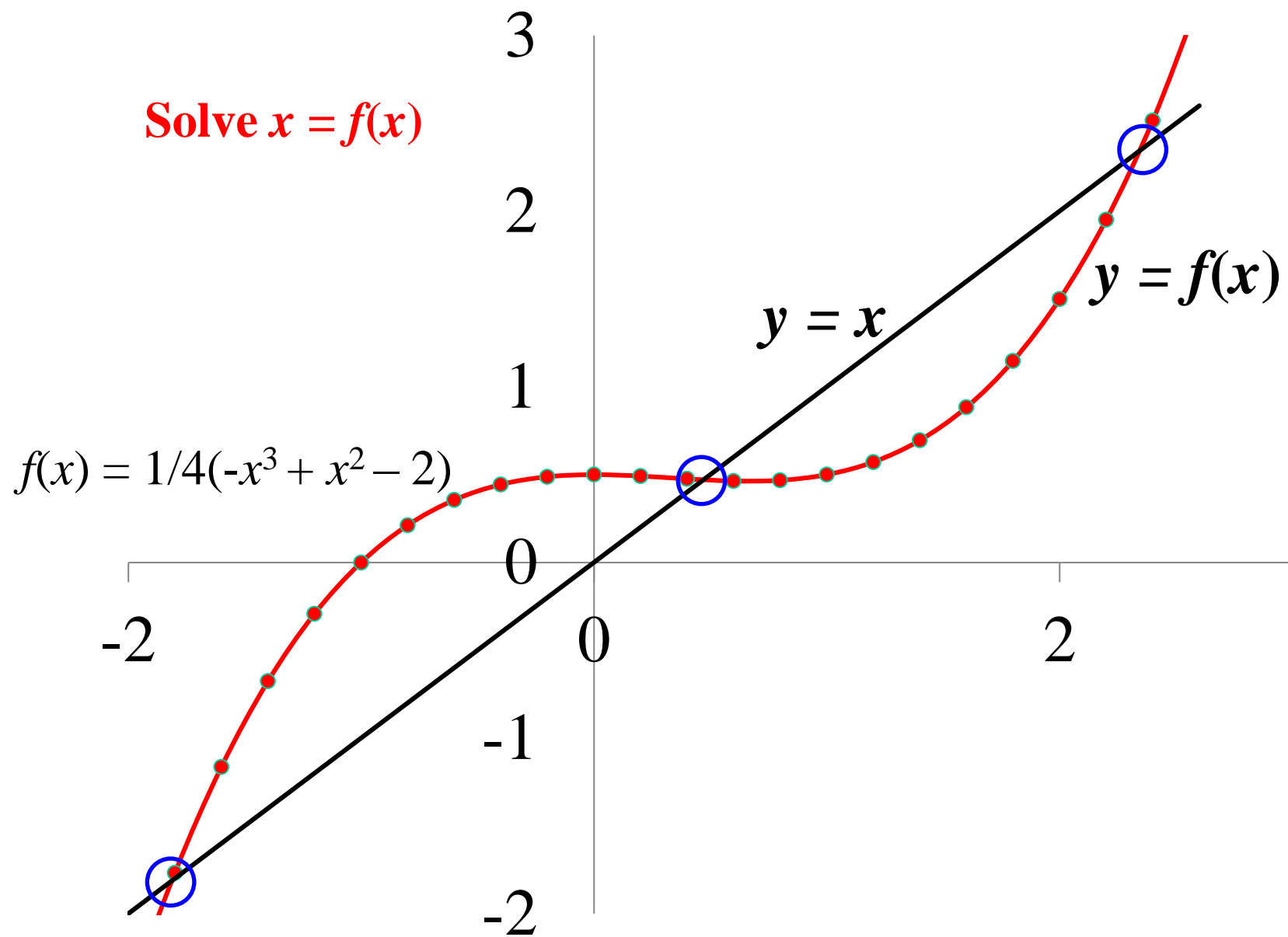
Mixing factor (混合係数) k_{mix} : Stabilize convergence

Initial value x_0

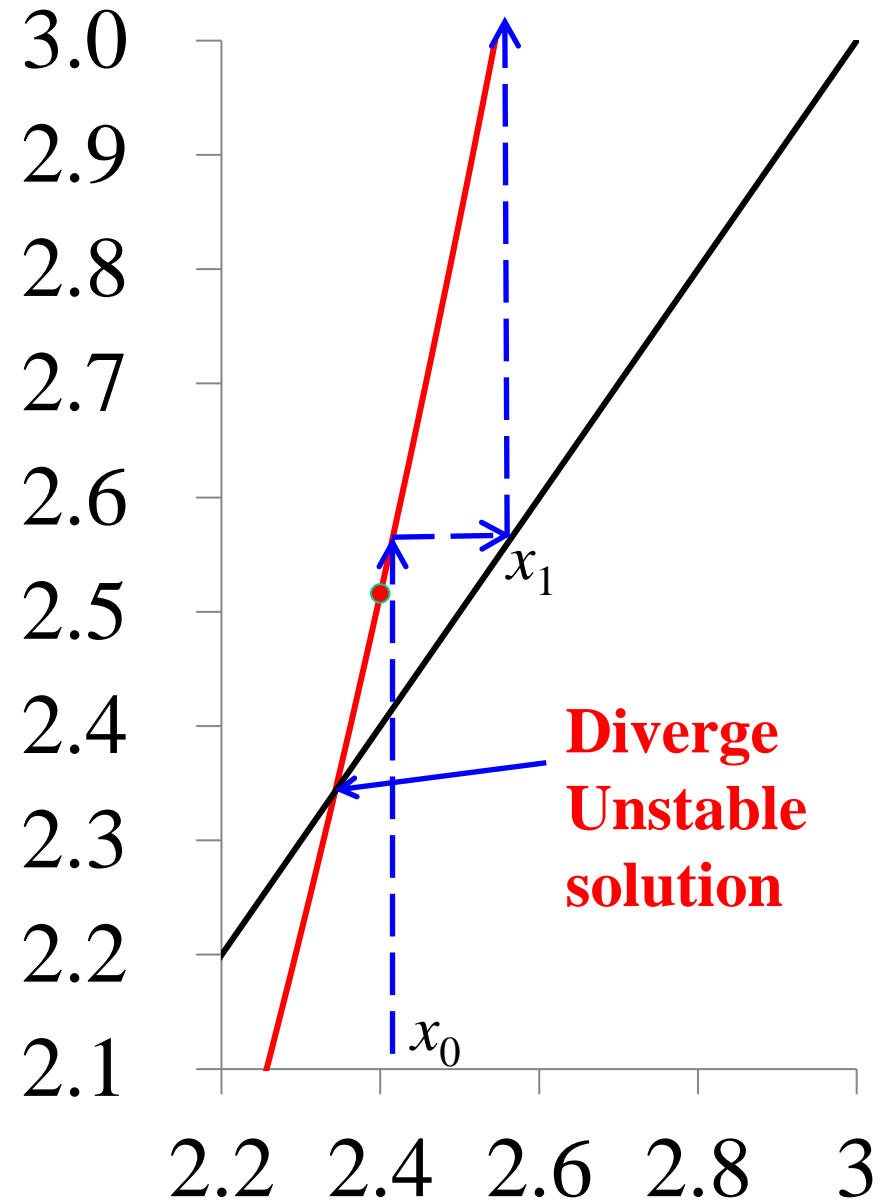
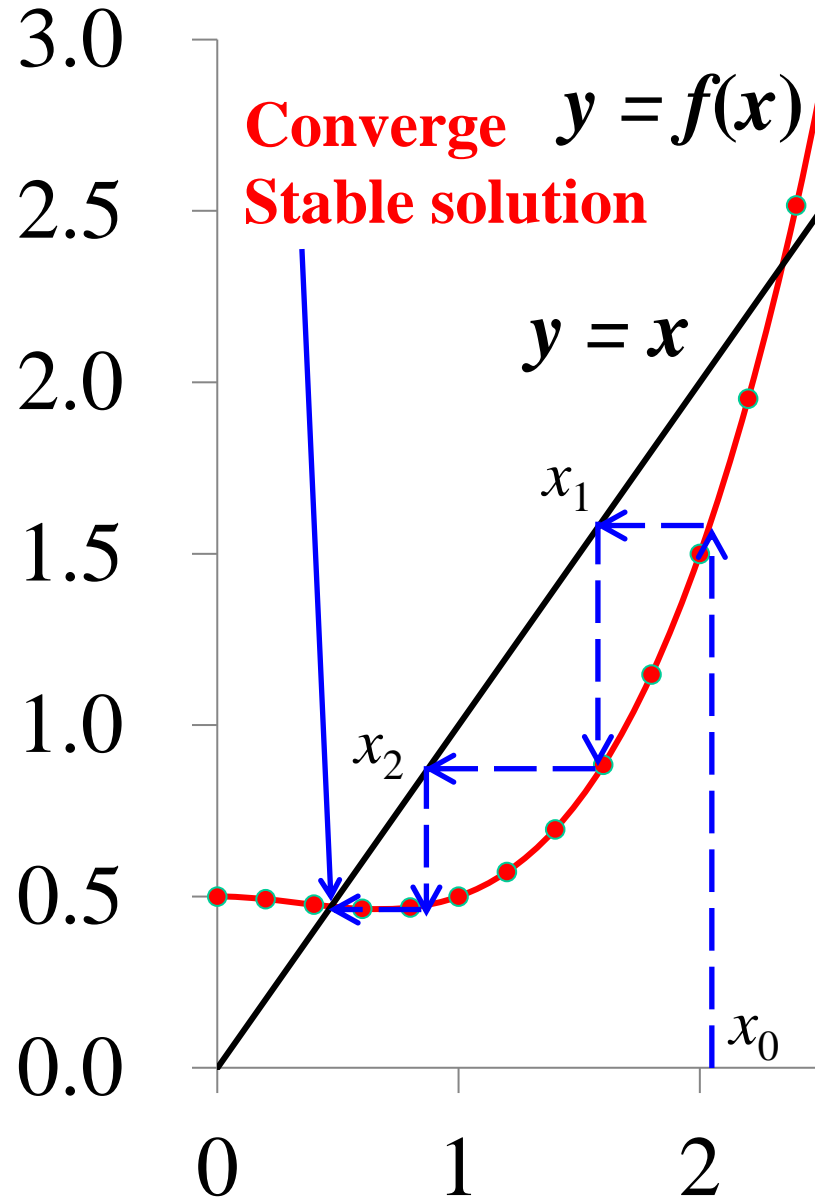
1st iteration : $x_1 = f(x_0) \Rightarrow x_1' = (1 - k_{\text{mix}}) x_0 + k_{\text{mix}} x_1$

2nd iteration: $x_2 = f(x_1') \dots$

Illustrative explanation of SC



SC: Convergence process



$f'(x) < 1$ must be satisfied for convergence

Example of SC: Diode with series resistance

$$I = I_0 \left[\exp \left(\frac{e}{nkT} (V - RI) \right) - 1 \right]$$

Repeat

$$I_i = I_0 \left[\exp \left(\frac{e}{nkT} (V - RI_{i-1}) \right) - 1 \right]$$

until $\text{abs}(I_i - I_{i-1}) < \text{EPS}$ is achieved

- E.g., initial voltages would be chosen as $V/2$ for the diode and the R
- This SC is not so stable; mixing factor k should be adjusted

For sequential calculations of $I - V$ characteristic, e.g., V from 0.0 to 1.0, using a preconverged result for the initial value of the next V will enhance convergence.

例えば V を順次変えて $I - V$ 特性を計算するような場合、すでに収束した値を次の V における初期値として利用すると早く収束できる。

SC-Diode.xlsx

i	I	Ical	error	I0=	1.E-12	A
0	2	-1E-12	2	n=	1	
1	1.8	-1E-12	1.8	T=	300	K
2	1.62	-1E-12	1.62	R=	1	ohm
3	1.458	-1E-12	1.458	V=	1	
4	1.3122	-1E-12	1.3122			
5	1.18098	-1E-12	1.18098	k=	0.1	
6	1.062882	-9.1E-13	1.06288			
7	0.956594	4.31E-12	0.95659			
8	0.860934	2.09E-10	0.86093			
9	0.774841	5.77E-09	0.77484			
10	0.697357	1.14E-07	0.69736			
11	0.627621	1.66E-06	0.62762			
12	0.564859	1.86E-05	0.56484			
13	0.508375	0.000163	0.50821			
14	0.457554	0.00115	0.4564			
15	0.411914	0.006655	0.40526			
16	0.371388	0.031631	0.33976			
17	0.337412	0.116849	0.22056			
18	0.315356	0.272927	0.04243			
19	0.311113	0.321305	0.01019			
20	0.312132	0.308953	0.00318			
21	0.311814	0.312754	0.00094			
22	0.311908	0.311626	0.00028			
23	0.31188	0.311965	8.5E-05			
24	0.311888	0.311863	2.5E-05			
25	0.311886	0.311893	7.6E-06			
26	0.311887	0.311884	2.3E-06			

First-principles calculation:

Self-consistent field (SCF, 自己無撞着) calculation

- Hamiltonian of one-electron quantum equation includes wave functions

$$\left\{ -\frac{1}{2} \nabla_l^2 - \sum_m \frac{Z_m}{r_{lm}} + \sum_m \int \frac{\rho_m(\mathbf{r}_m)}{r_{lm}} d\mathbf{r}_m + V_{xl}(\mathbf{r}_l) \right\} \phi_l(\mathbf{r}_l) = \varepsilon_l \phi_l(\mathbf{r}_l)$$

- First-step calculation requires electron density guessed / assumed ρ_{ini} :
e.g., by uniform density, sum of atomic electron density,,



- Electron density ρ_{fin} is calculated the solved wave functions, but ρ_{fin} would be different from ρ_{ini}



ρ_{ini} must be equal to ρ_{fin} , otherwise
these loss physical meaning

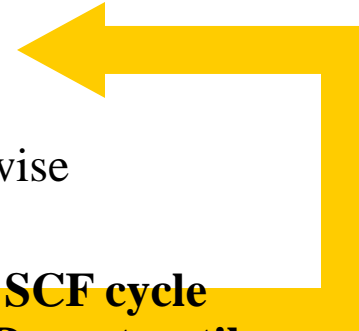
- More appropriate ρ_{new} is guessed from ρ_{fin} and ρ_{ini} ,
and repete the above calculations

ex.: $\rho_{\text{new}} = \rho_{\text{ini}} + k_{\text{mix}}(\rho_{\text{fin}} + \rho_{\text{ini}})$

k_{mix} : **Mixing factor**

A parameter to suppress divergence of the SCF calculation
close to 1 would be easily diverged, close to 0 causes slow convergence

SCF cycle
Repeat until $\rho_{\text{fin}} = \rho_{\text{ini}}$



Example: SCF/structure relaxation by VASP

```
tkamiya@csrv0:~/Work/LaCrAsO/SpinPolarized
ファイル(E) 編集(E) 表示(V) 端末(T) タブ(B) ヘルプ(H)
1 F= -.24922201E+03 E0= -.24922201E+03 d E =-.249222E+03 mag= 17.6753
curvature: 0.00 expect dE= 0.000E+00 dE for cont linesearch 0.000E+00
trial: gam= 0.00000 g(F)= 0.620E+00 g(S)= 0.305E-01 ort = 0.000E+00 (trialstep = 0.100E+01
)
search vector abs. value= 0.650E+00
bond charge predicted
      N      E      dE      d eps      ncg      rms      rms(c)
DAV: 1 -0.249256423264E+03 -0.24926E+03 -0.54781E+01 3528 0.200E+01 0.196E+00
DAV: 2 -0.249670978228E+03 -0.41455E+00 -0.52988E+00 4416 0.955E+00 0.161E+00
DAV: 3 -0.249672461360E+03 -0.14831E-02 -0.53814E-01 4640 0.336E+00 0.153E+00
DAV: 4 -0.249667045995E+03 0.54154E-02 -0.45192E-01 4632 0.183E+00 0.129E+00
DAV: 5 -0.249662986402E+03 0.40596E-02 -0.16171E-01 4664 0.134E+00 0.113E+00
DAV: 6 -0.249664501455E+03 -0.15151E-02 -0.86520E-02 4520 0.152E+00 0.943E-01
DAV: 7 -0.249658663938E+03 0.58375E-02 -0.36669E-02 4626 0.103E+00 0.315E-01
DAV: 8 -0.249657255947E+03 0.14080E-02 -0.11030E-02 4432 0.529E-01 0.406E-01
DAV: 9 -0.249656661683E+03 0.59426E-03 -0.64937E-03 3424 0.480E-01 0.219E-01
DAV: 10 -0.249654538004E+03 0.21237E-02 -0.11755E-03 2528 0.225E-01 0.151E-01
DAV: 11 -0.249654612437E+03 -0.74432E-04 -0.11566E-03 2520 0.213E-01
2 F= -.24965461E+03 E0= -.24965461E+03 d E =-.432599E+00 mag= 18.2912
trial-energy change: -0.432599 1 .order -0.416777 -0.650072 -0.183481
step: 1.3105(harm= 1.3932) dis= 0.06748 next Energy= -249.683568 (dE=-0.462E+00)
bond charge predicted
      N      E      dE      d eps      ncg      rms      rms(c)
DAV: 1 -0.249658788237E+03 -0.24966E+03 -0.53760E+00 3536 0.623E+00 0.599E-01
DAV: 2 -0.249698102900E+03 -0.39315E-01 -0.48908E-01 4528 0.303E+00 0.671E-01
```

Typical iteration of SC calculation

Find the solution of $f(x, \rho(x)) = 0$:

Case this is easily done if $\rho(x)$ is provided

1. Assume $\rho(x)$ and solve $f(x, \rho(x)) = 0$ to get approximate x_i
2. Calculate $\rho(x_i)$ with the obtained x_i , solve $f(x, \rho(x_i)) = 0$, and get improved approximation x_{i+1}
3. Repeat 1 – 2 so as to decrease $|\rho(x_{i+1}) - \rho(x_i)|$, $|x_{i+1} - x_i|$ to required accuracy

Self-consistent approach (自己無動着計算)

May be diverged if the obtained x_i' is used for x_{i+1}

=> **Stabilize convergece using mixing factor** (混合係数) k_{mix}

Initial x_0

First iteration: $x_1 = f(x_0)$ => $x_1' = (1 - k_{\text{mix}}) x_0 + k_{\text{mix}} x_1$

Next iteration: $x_2 = f(x_1')$

Problems of SC calculations

- **Some solutions would not be obtained** (収束しない解があり得る)
 $f'(x) < 1$ must be satisfied at the solution to obtain the solution of $x = f(x)$
=> Conversion of the equation may help, but not always
- **Convergence is not stable**
mixing factor may improve

For many cases, use another method such as Newton method

- **Cases SC method is effective**
Initial values close to the solution
Effect of SC parameters is small to the equation
(自己無撞着変数の方程式への影響が小さい)
SC parameters have good convergence
(自己無撞着変数の収束特性が良く、予測できる場合)

How to solve equations?

More sophisticated algorithms

Newton-Raphson method

Solve $f(x) = 0$

Start from initial guess: x_0

$x_0 + dx$ is supposed to be a solution

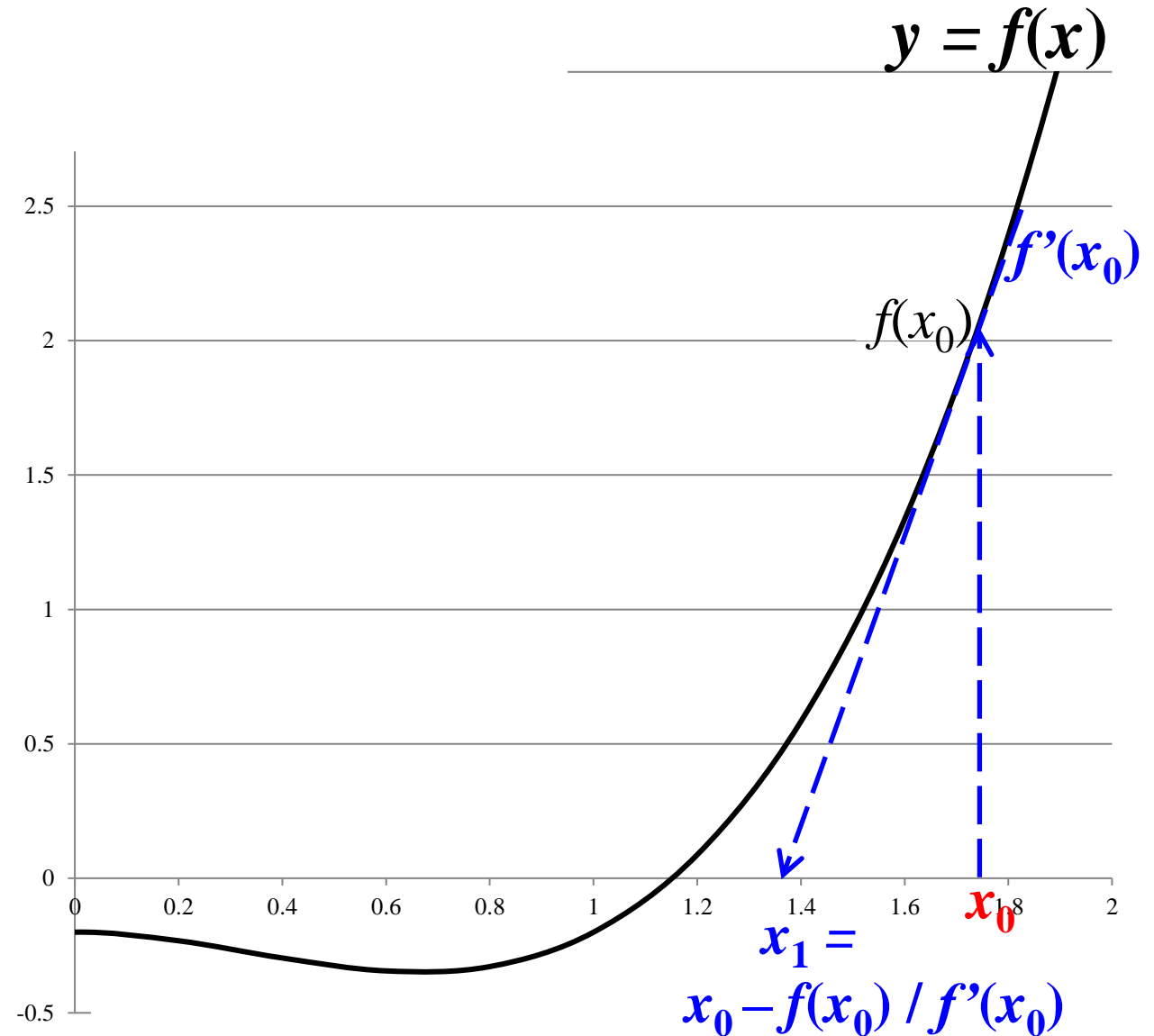
$$f(x_0 + dx) = f(x_0) + dx f'(x_0) \sim 0$$

$$\Rightarrow x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$$

Stabilize convergence:

$$x_{k+1} = x_k - f(x_k) / f'(x_k) / (1 + \lambda)$$

λ : Damping Factor



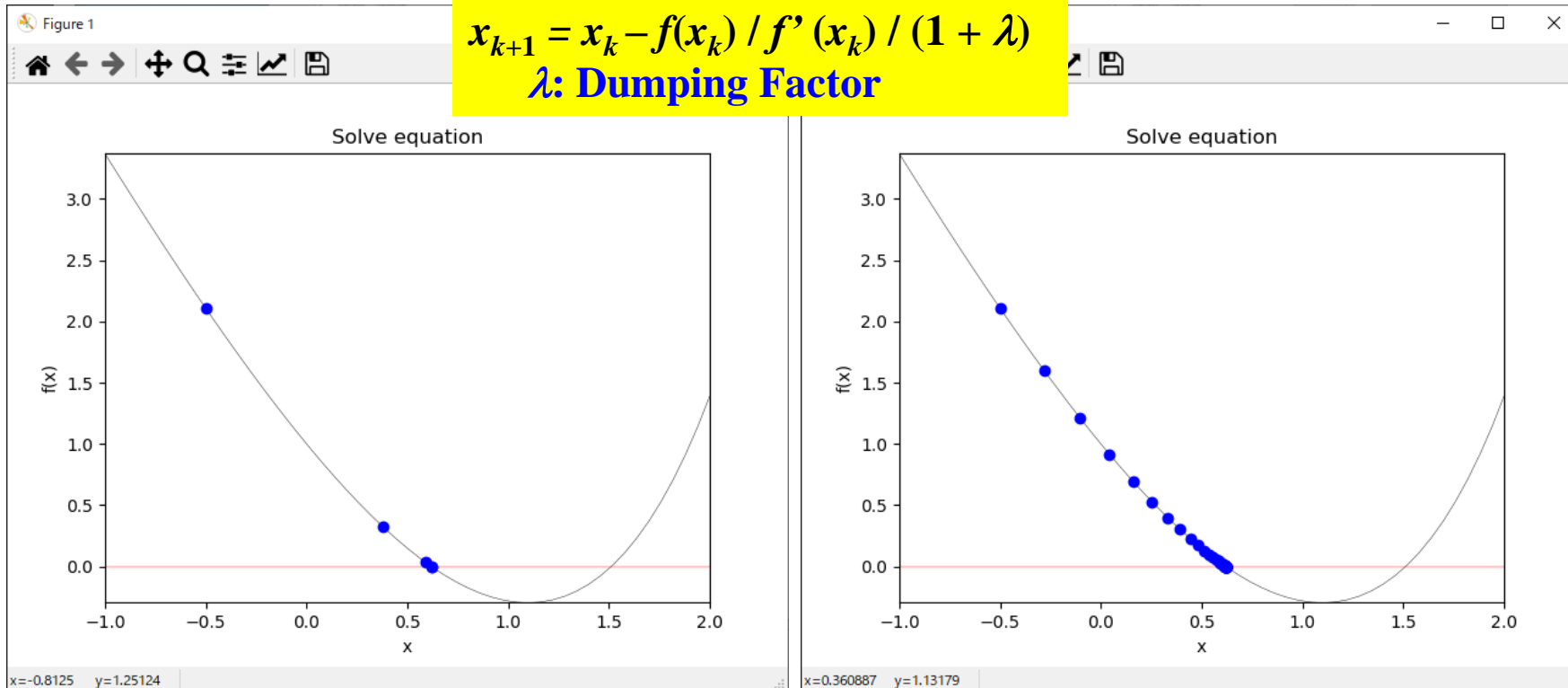
Program: equation-newton-Raphson.py

Usage: python equation-newton-raphson.py x0 dump t_{sleep}

$$f(x) = \exp(x) - 3.0x$$

python equation-newton-raphson.py -0.5 0

python equation-newton-raphson.py -0.5 3



Effect of dumping factor (収束過程の比較)

$$f(x) = \exp(x) - 3x = 0 \text{ (initial } x = 0) \quad \text{Exact } 0.619061$$

Newton-Raphson (Dumping factor = 0)

Iter.	x	$ x_i - x_{i-1} $
1	0.5	
2	0.610059654958962	0.110059654958962
3	0.61899677974154	0.00893712478257794
4	0.619061283355313	6.4503613773092e-005
5	0.619061286735945	3.38063244722622e-009
6	0.619061286735945	-1.94296000199483e-016

Newton-Raphson (Dumping factor = 0.1)

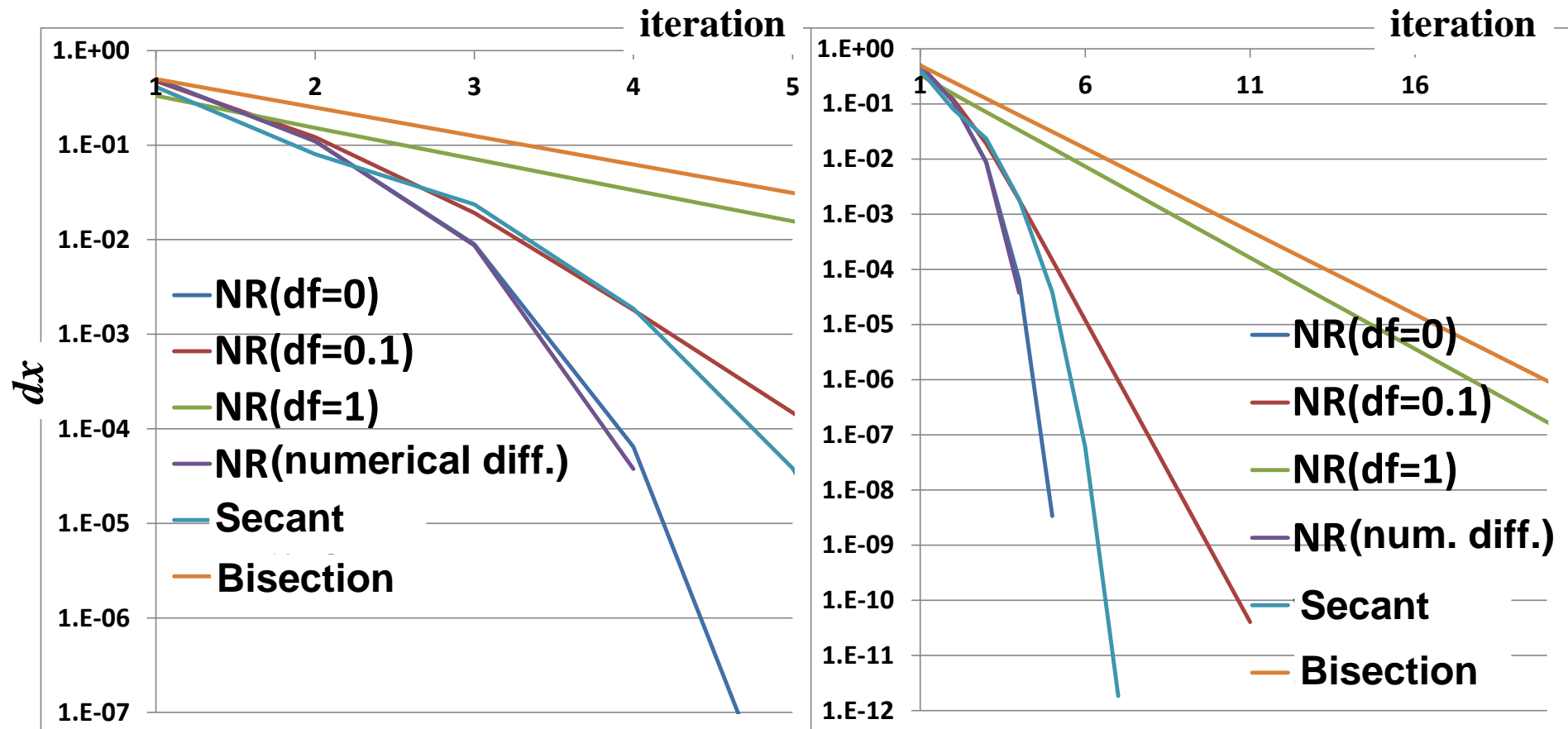
1	0.476190476190476	
2	0.597901649246081	0.121711173055605
3	0.617090542717403	0.0191888934713221
4	0.618900291486661	0.00180974876925825
5	0.619048316423879	0.000148024937217564
6	0.619060243007723	1.19265838440254e-005
7	0.619061202754359	9.59746635487409e-007
8	0.619061279978579	7.72242198569211e-008
9	0.619061286192231	6.21365241490959e-009
10	0.619061286692197	4.99965669237101e-010
11	0.619061286732425	4.0228535713285e-011

Newton-Raphson (Dumping factor = 1.0)

1	0.333333333333333	
2	0.485235618882813	0.15190228554948
3	0.556317491275292	0.0710818723924794
4	0.589692022113926	0.0333745308386341
5	0.605333177012923	0.0156411548989961
6	0.612649553494255	0.00731637648133212
7	0.616067929129785	0.00341837563553035
8	0.617664103982484	0.00159617485269905
9	0.618409199563502	0.00074509558101794
10	0.618756961315507	0.000347761752005284
11	0.618919262817103	0.000162301501596124

Effect of dumping factor: Convergence process

$f(x) = \exp(x) - 3x = 0$ (initial $x = 0$) Exact 0.619061



$$x_{k+1} = x_k - f(x_k) / (f'(x_k) + \lambda)$$

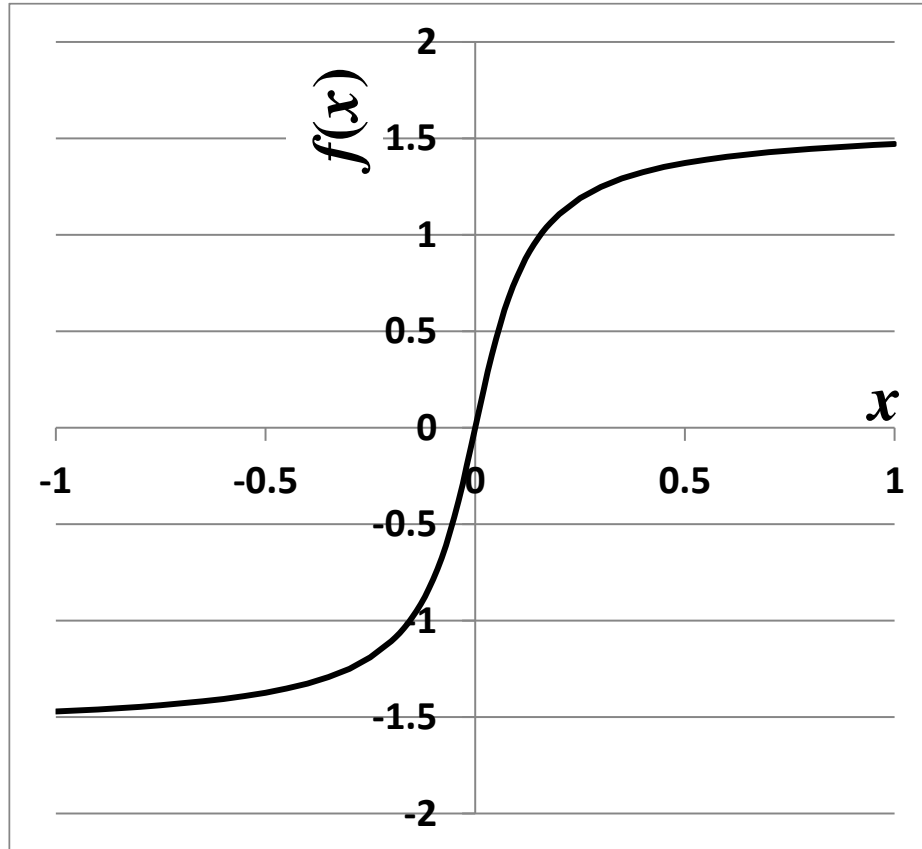
 λ : Dumping Factor

NR: Newton-Raphson method
df: Dumping Factor

Case Newton method succeeds

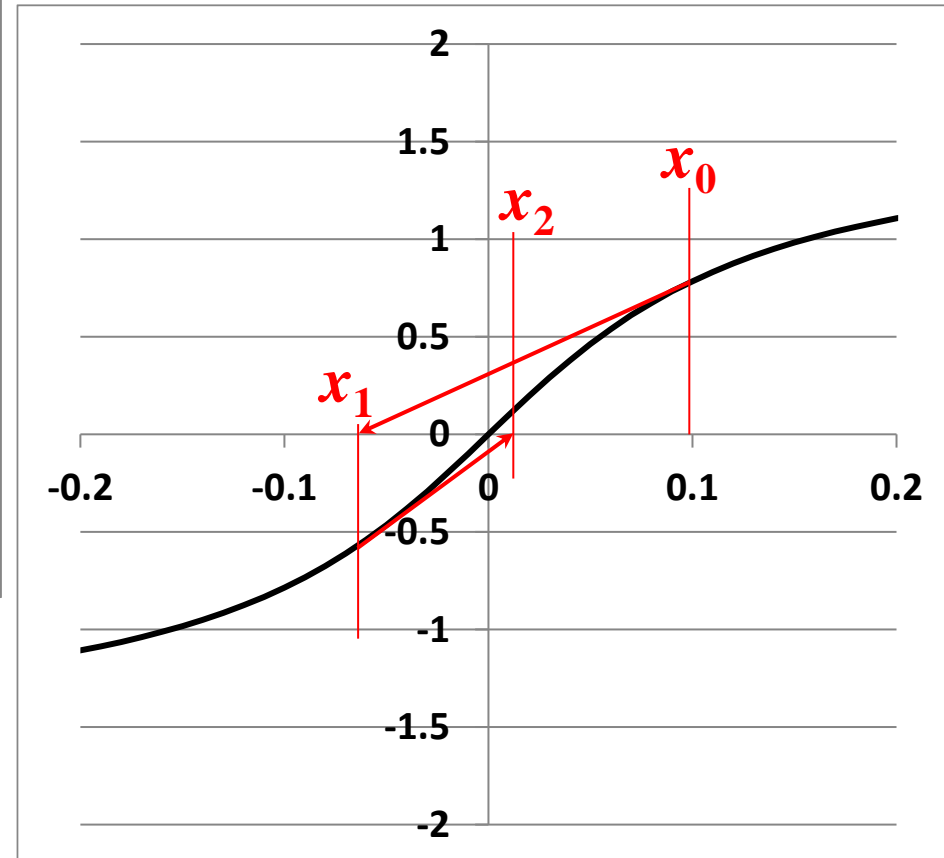
$$f(x) = \tan^{-1}(10x)$$

initial $x = 0.1$



A case to reach convergence

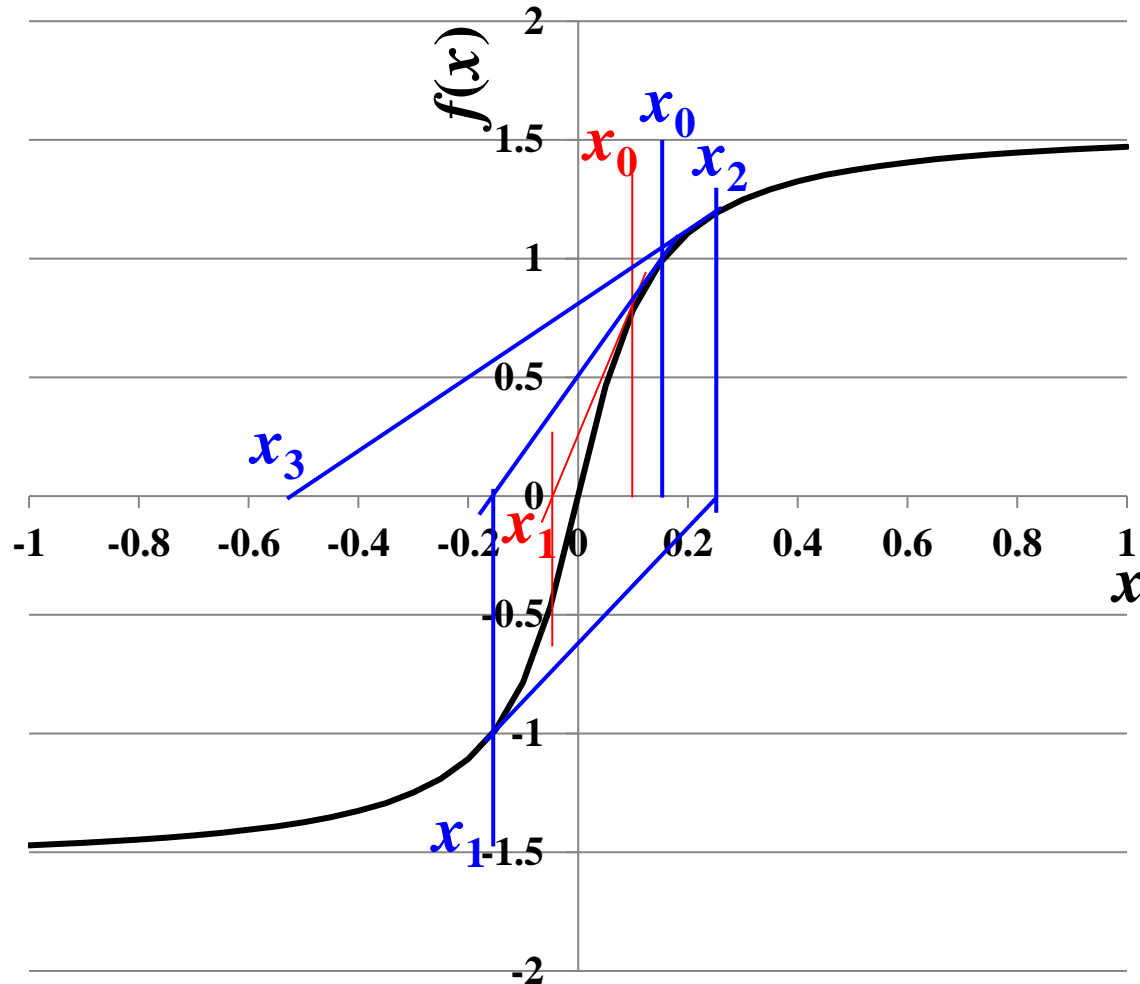
i	x	f(x)	df/dx	dx
0	0.1	0.7854	5	-0.1571
1	-0.05708	-0.5187	7.54257	0.06877
2	0.011686	0.11633	9.86527	-0.0118
3	-0.00011	-0.0011	9.99999	0.00011
4	1.15E-10	1.2E-09	10	-1E-10



Case Newton method fails

$$f(x) = \tan^{-1}(10x)$$

initial $x = 0.15$



Diverged ($\lambda = 0$)

i	x	f(x)	df/dx	dx
0	0.15	0.98279	3.07692	-0.3194
1	-0.16941	-1.0375	2.58404	0.40152
2	0.232112	1.164	1.56553	-0.7435
3	-0.51141	-1.3777	0.36827	3.74095
4	3.229546	1.53984	0.00958	-160.76
5	-157.529	-1.5702	4E-06	389644
6	389486.7	1.5708	1.1E-12	-1E+12

$$x_{k+1} = x_k - f(x_k) / (f'(x_k) + \lambda)$$

λ : Dumping Factor

**Stabilize convergence
by choosing $\lambda(\lambda = 1)$**

i	x	f(x)	df/dx	dx
0	0.15	0.98279	3.07692	-0.2411
1	-0.09106	-0.7387	5.46675	0.11422
2	0.023161	0.2276	9.49088	-0.0217
3	0.001466	0.01466	9.99785	-0.0013
4	0.000133	0.00133	9.99998	-0.0001
5	1.21E-05	0.00012	10	-1E-05
6	1.1E-06	1.1E-05	10	-1E-06
7	1E-07	1E-06	10	-9E-08
8	9.09E-09	9.1E-08	10	-8E-09
9	8.27E-10	8.3E-09	10	-8E-10

Program: Electron density in metal

Issues for integrating $N(e)f(e)$

- Wide integration range $E = 0 \sim E_F + \alpha k_B T$ – several eV (accuracy at the order of $\exp(-\alpha)$)
 - Important range for accuracy is the range of $\alpha k_B T \sim 0.1$ eV around E_F
 - For numerical integration, E mesh ΔE should be very small around E_F (if $0.01\alpha k_B T$, $\Delta E \sim 1$ meV)
=> Not good to use the same ΔE for the whole integration range $E = 0 \sim E_F + \alpha k_B T$
- => ▪ **Divide integration range** (Analytical integration may be employed for $0 \sim E_F - \alpha k_B T$)
- **Better to employ accuracy-guaranteed library for integration**
`python integrate.quad()` can accept accuracy as `epsrel` variable

Program: N-integration-metal.py

Ex.: `python N-integration-metal.py 300 5.0`

At 300 K, $E_F = 5.0$ eV

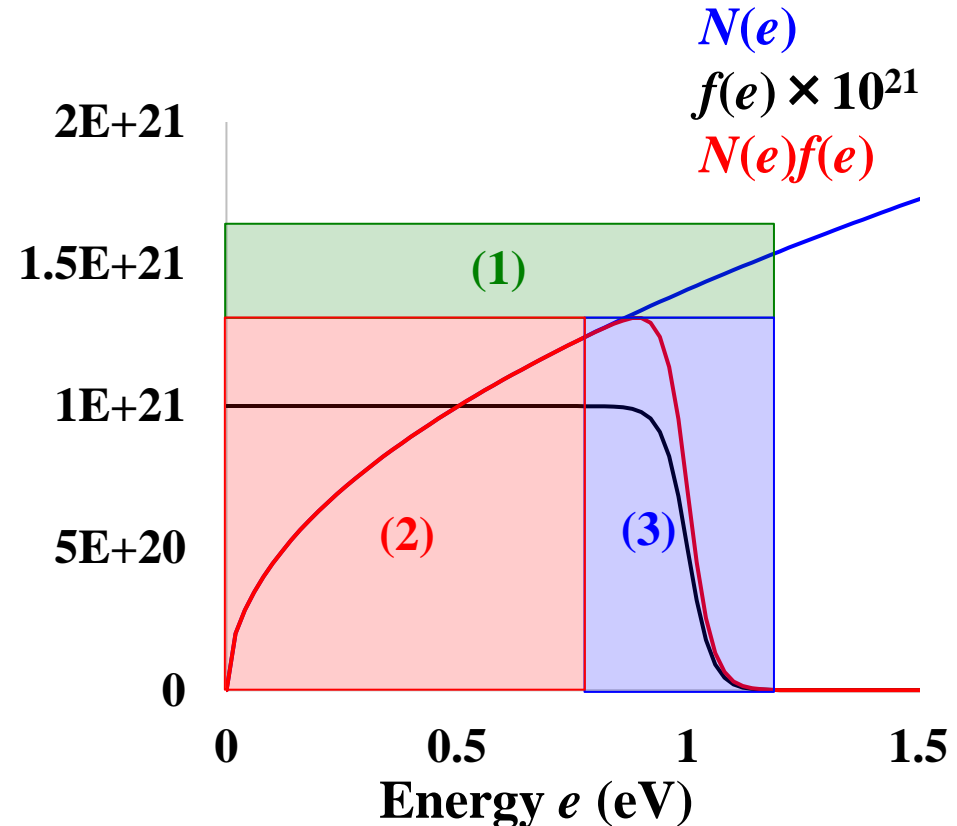
Time is measured for 300 cycles calculation

8 digit accuracy (`epsrel = 1e-8`), $\alpha = 6$:

range	time (300 cycles)
(1) $0 \sim E_F + \alpha k_B T$	0.109 s
(2) $0 \sim E_F - \alpha k_B T$	0.063 s
(3) $E_F - \alpha k_B T \sim E_F + \alpha k_B T$	0.016 s

(2) + (3) is faster by ~30 % than (1).

Employing analytical integration for (2)
is faster by a factor of 10



Program: T dependence of E_F for metal

$E_F(T)$ is determined by $N_e = \int N(e)f(e, E_F)de$ for the given electron number N_e

$N(e)f(e, E_F)$ is integrated in the range $E = 0 - \infty$ (actually up to $E_F + \alpha k_B T$)

The initial value of $E_F(T)$ can be taken as the analytical form of $E_F(0)$ at 0 K.

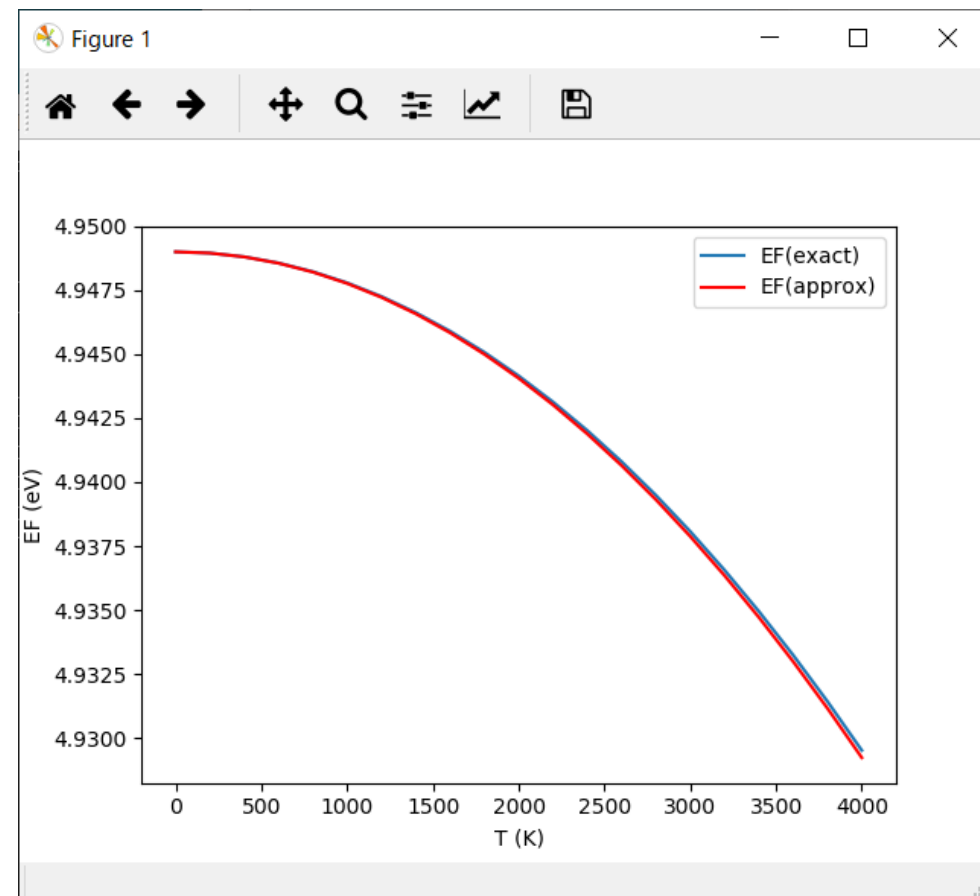
Since the variation of $E_F(T)$ is small, the Newton method stability converges.

Compare with the approx. form $E_F(T) = E_F(0) - \frac{\pi^2}{6} (k_B T)^2 N'(E_F(0)) / N(E_F(0))$

Program: EF-T-metal.py

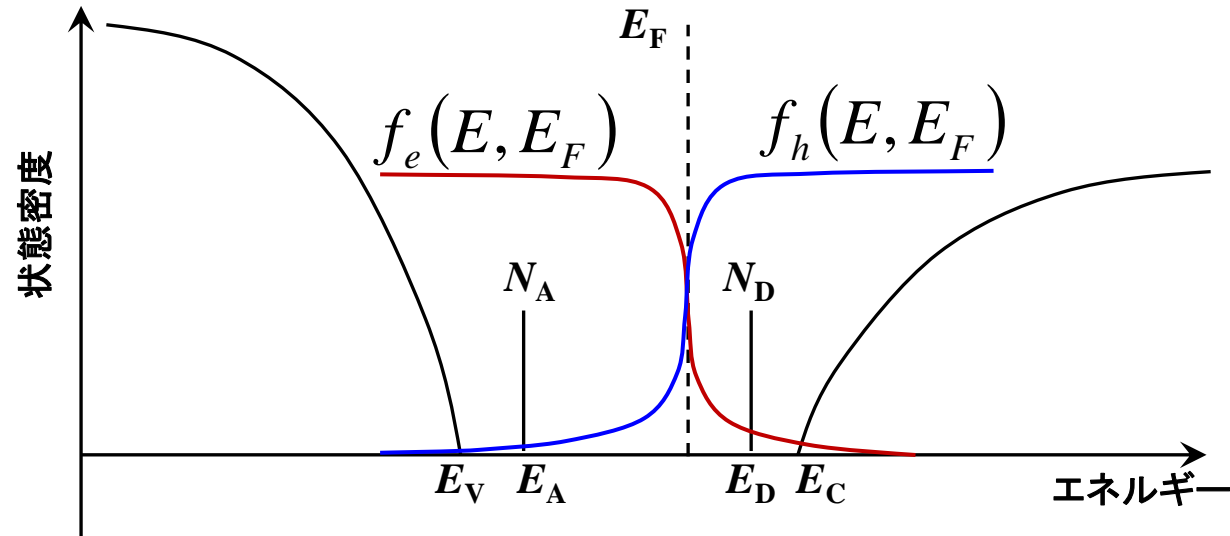
Ex.: `python EF-T-metal.py`

T (K)	E_F (Newton, eV)	E_F (approx., eV)
0	4.948988	4.948988
600	4.948554	4.948544
1200	4.947248	4.947211
1800	4.945069	4.944990
2400	4.942013	4.941880
3000	4.938075	4.937882
3600	4.933247	4.932994
4000	4.929529	4.929243



Density of states, n_e , and n_h in semiconductor

Total density of states: $D(E) = D_e(E) + D_h(E) + D_D(E) + D_A(E)$



Valence band

$$D_h(E) = D_{V0} \sqrt{E_V - E}$$

$$D_A(E) = N_A \delta(E - E_A)$$

$$f_h(E, E_F) = \frac{1}{\exp(\beta(E_F - E)) + 1}$$

Free hole density

$$n_h = \int_{-\infty}^{E_V} f_h(E, E_F) D_h(E) dE$$

Ionized acceptor density

$$N_A^- = N_A (1 - f_h(E_A, E_F))$$

Conduction band

$$D_e(E) = D_{C0} \sqrt{E - E_C}$$

$$D_D(E) = N_D \delta(E - E_D)$$

$$f_e(E, E_F) = \frac{1}{\exp(\beta(E - E_F)) + 1}$$

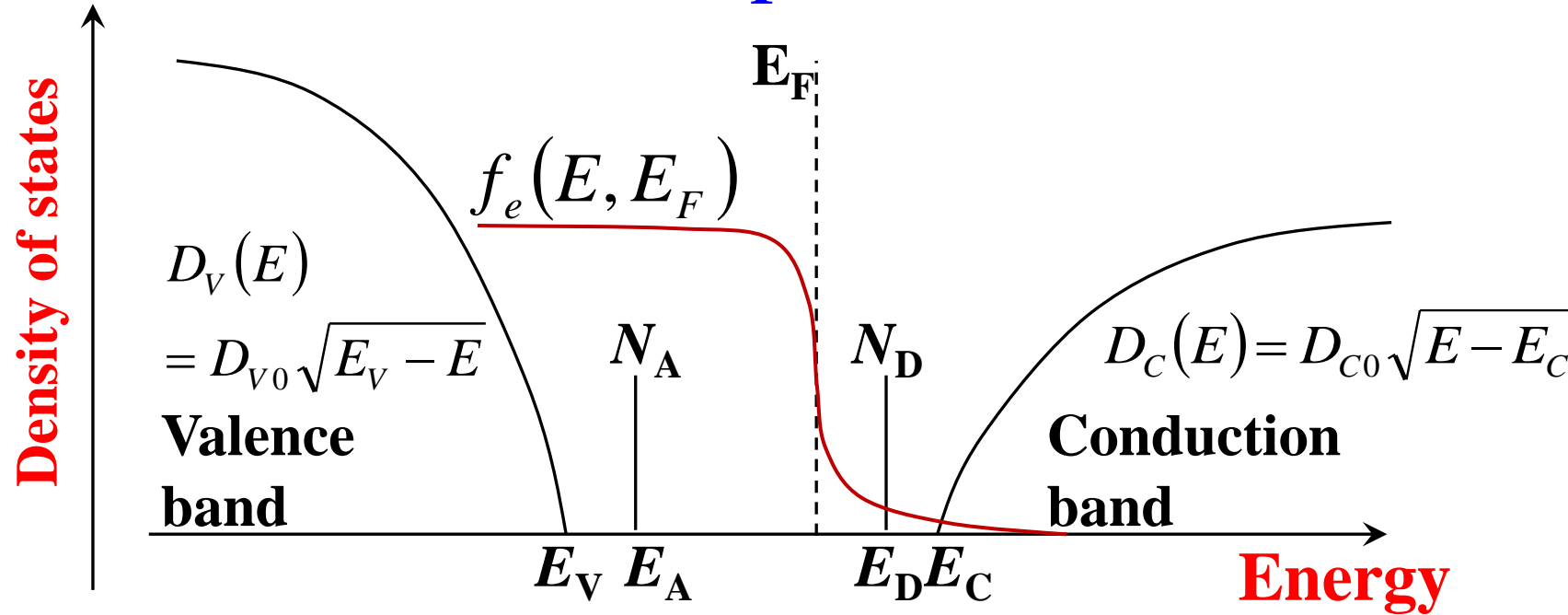
Free electron density

$$n_e = \int_{E_C}^{\infty} f_e(E) D_e(E) dE$$

Ionized donor density

$$N_D^+ = N_D (1 - f_e(E_D, E_F))$$

How to determine E_F for semiconductors



Charge neutrality condition

$$N_A^- + N_e = N_D^+ + N_h \quad \longrightarrow \quad E_F$$

$$N_e = \int_{E_C}^{\infty} D_C(E) f_e(E, E_F) dE$$

$$N_D^+ = N_D [1 - f_e(E_D, E_F)]$$

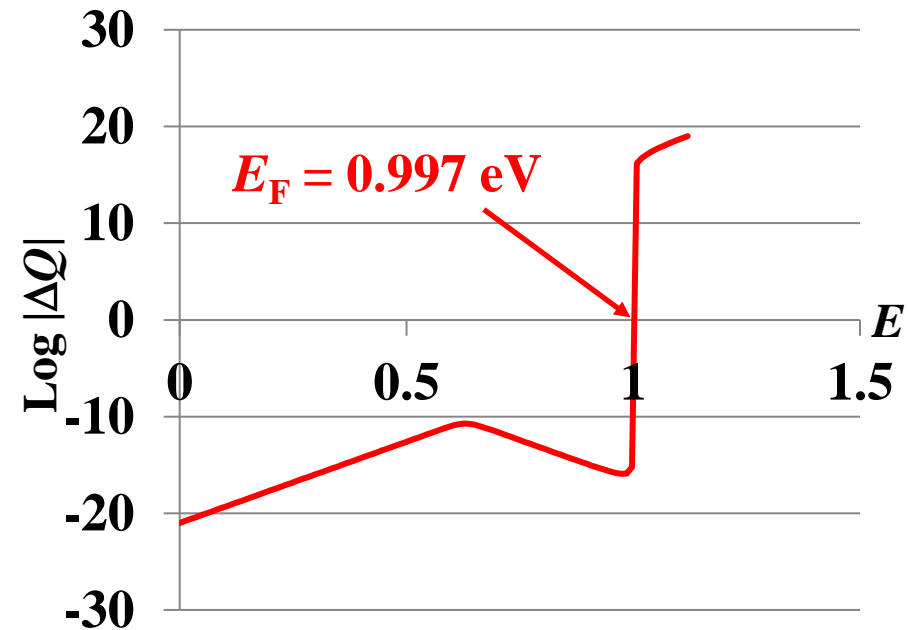
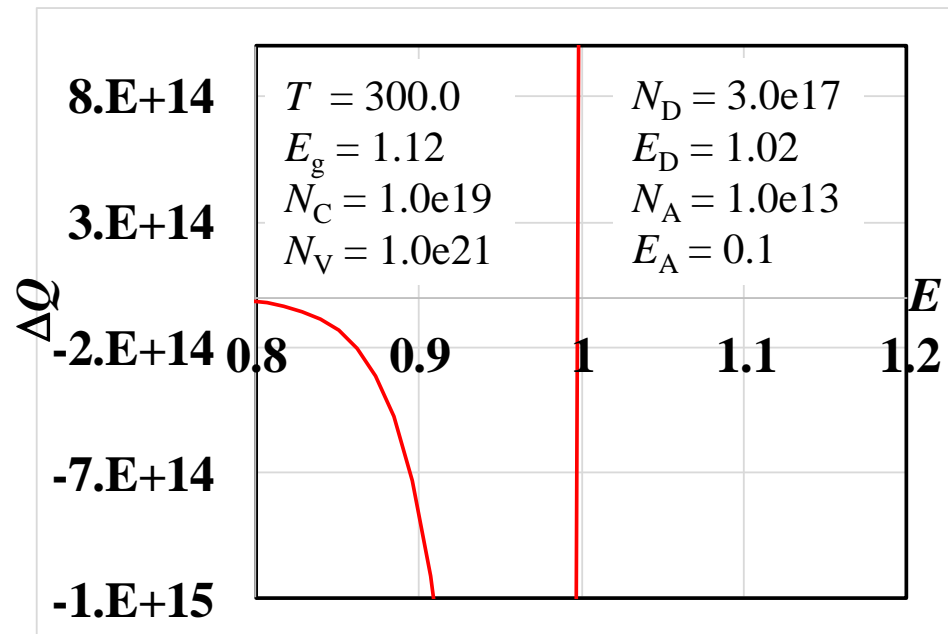
How to calculate E_F : Illustrative solution

$$N_e = \int_{E_C}^{\infty} D_C(E) f_e(E, E_F) dE \quad N_h = \int_{E_C}^{\infty} D_V(E) f_h(E, E_F) dE$$

$$N_D^+ = N_D [1 - f_e(E_D, E_F)] \quad N_A^- = N_A [1 - f_h(E_A, E_F)]$$

$$f_h(E, E_F) = 1 - f_e(E, E_F)$$

Plot $\Delta Q = (N_A^- + N_e) - (N_D^+ + N_h)$ w.r.t. E_F and find $\Delta Q = 0$



Bisection method (二分法): Continuous func (連続関数)

Solution of $f(x) = 0$ for (monotonic) continuous function $f(x)$

1. Start from a range $[x_0, x_1]$ where $f(x_0) < 0$ & $f(x_1) > 0$
(or $f(x_0) > 0$ & $f(x_1) < 0$)

*** Solution exist in this range for a monotonic function**

2. Solve the equation by the following iterative procedure

Case $f(x_0) < 0$ and $f(x_1) > 0$: Judge by $f(x_0) \cdot f(x_1) < 0$

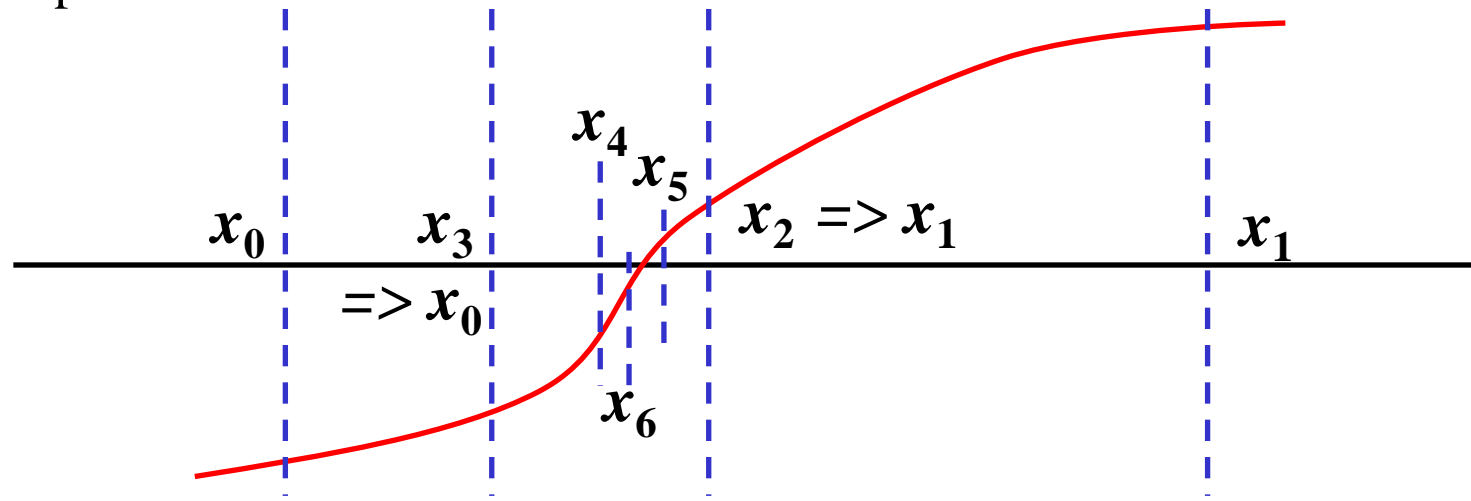
1. $x_2 = (x_0 + x_1) / 2.0$

2. If $f(x_2) > 0$ ($f(x_0) \cdot f(x_2) < 0$), x_1 is replaced with x_2

If $f(x_2) < 0$ ($f(x_1) \cdot f(x_2) < 0$), x_0 is replaced with x_2

3. Solution x_2 is obtained when $|x_1 - x_0|$, $|f(x_1) - f(x_0)|$ becomes less than EPS.

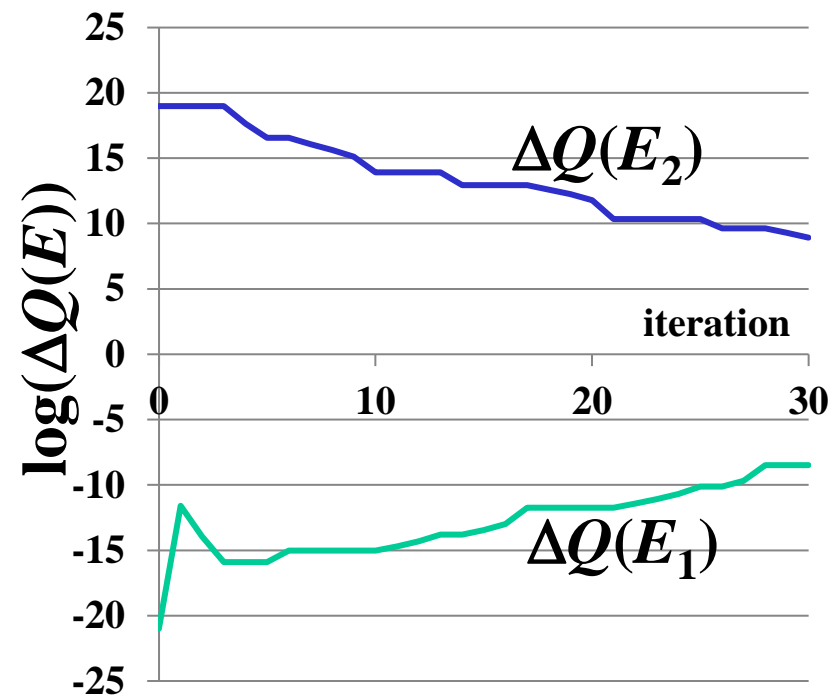
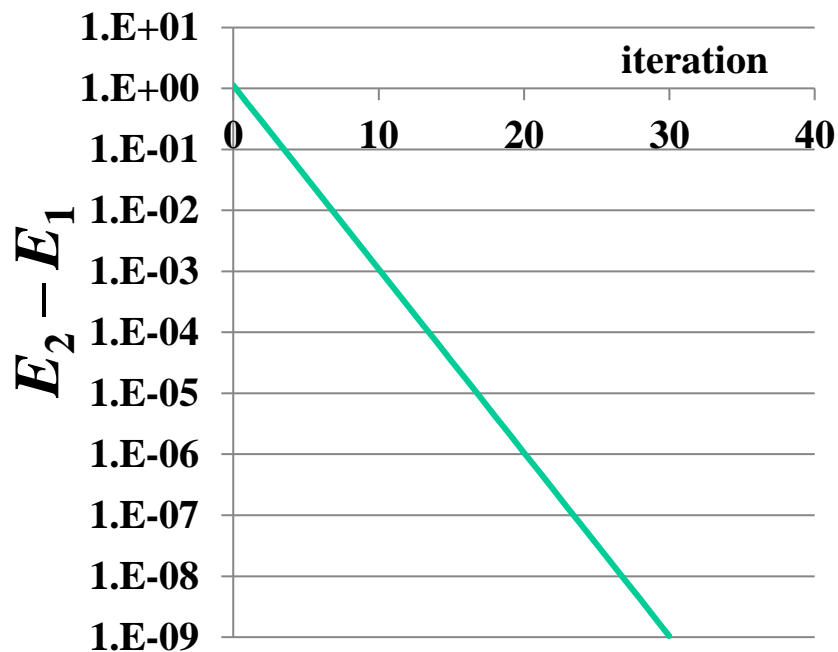
4. Repet 1 – 3



E_F by bisection method: Convergence procedure

Initial range: $[E_1, E_2] = [E_V = 0, E_C = E_g]$

Find $\Delta Q = (N_A^- + N_e) - (N_D^+ + N_h) = 0$



After 30 times iterations

$E_F = [0.9985173589, 0.9985173599]$

$dQ = [-3 \times 10^8, 8 \times 10^8]$

Program: EF-T-semiconductor.py

Program: EF-T-semiconductor.py

Usage: `python EF-T-semiconductor.py EA NA ED ND Ec Nv Nc`

Ex.: `python EF-T-semiconductor.py 0.05 1.0e15 0.95 1.0e16 1.0 1.2e19 2.1e18`

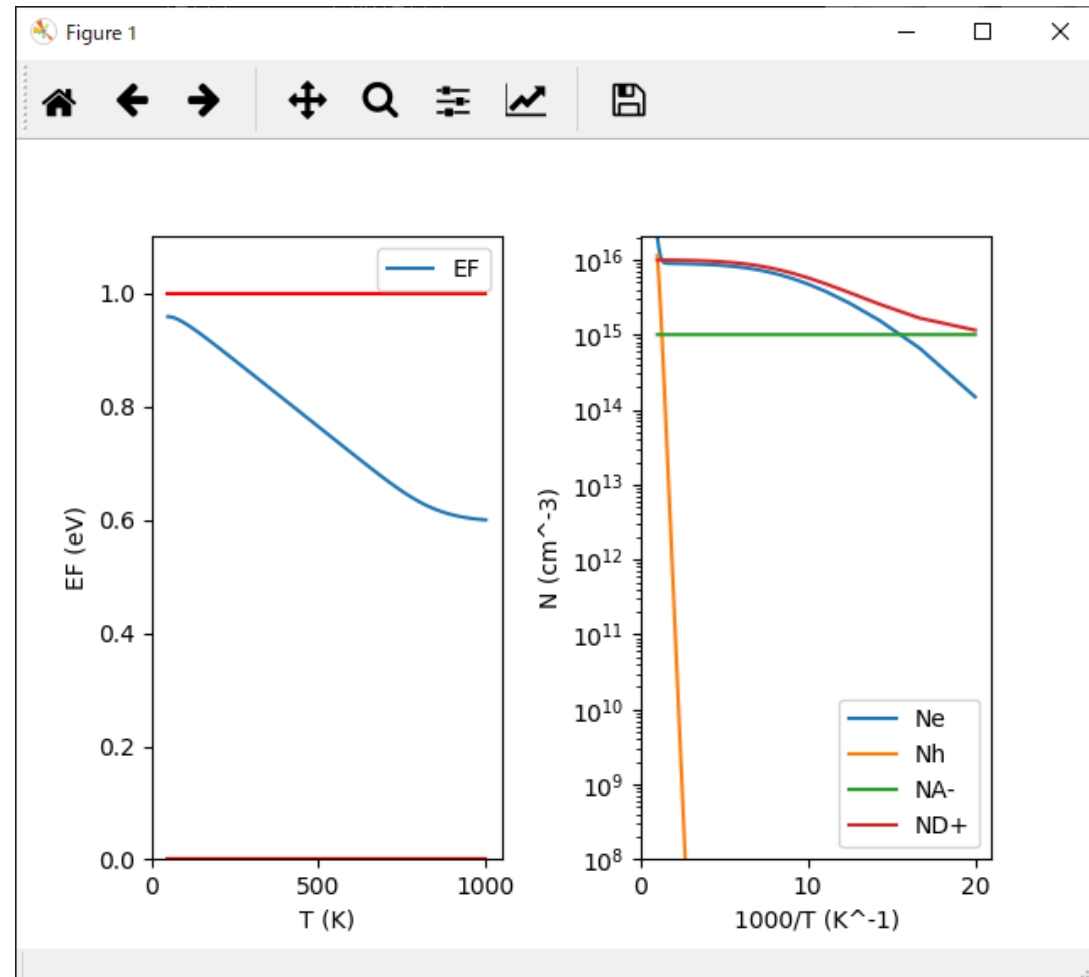
$E_c = 0$, $E_v = 1.0$ eV (= band gap)

$E_A = 0.05$ eV, $N_A = 10^{15}$ cm⁻³,

$E_D = 0.95$ eV, $N_D = 10^{16}$ cm⁻³

$N_c = 1.2 \times 10^{19}$ cm⁻³

$N_v = 2.1 \times 10^{18}$ cm⁻³



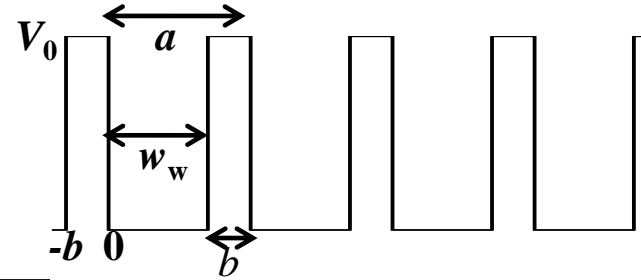
Multi-values equation: Kronig-Penney model

Solution of $\left(-\frac{\hbar^2}{2m} \frac{d^2}{dx^2} + V(x)\right) \phi = E\phi$

$\phi_k(x) = \exp(ikx)u(x), u(x+a) = u(x)$

In well: $\phi(x) = A \exp(i\alpha x) + B \exp(-i\alpha x) \quad \alpha = \sqrt{2mE} / \hbar$

In barrier: $\phi(x) = C \exp(\beta x) + D \exp(-\beta x) \quad \beta = \sqrt{2m(V_0 - E)} / \hbar$



Boundary condition: $\phi_k(x)$ and $\phi_k'(x)$ are continuous at $x = 0$ and $-b$

Bloch's theorem : $\phi_k(x + a) = \lambda \phi_k(x), \lambda = \exp(ika)$

$$\begin{pmatrix} 1 & 1 & -1 & -1 \\ i\alpha & -i\alpha & -\beta & \beta \\ \exp(i\alpha w_w) & \exp(-i\alpha w_w) & -\lambda \exp(-\beta b) & -\lambda \exp(\beta b) \\ i\alpha \exp(i\alpha w_w) & -i\alpha \exp(-i\alpha w_w) & -\beta \lambda \exp(-\beta b) & \beta \lambda \exp(\beta b) \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The determinant of the left matrix must be 0:

$$\cos ka = \left(\frac{\beta(E)^2 - \alpha(E)^2}{2\alpha(E)\beta(E)} \sin \alpha(E)w_w \sinh \beta(E)b + \cos \alpha(E)w_w \cosh \beta(E)b \right)$$

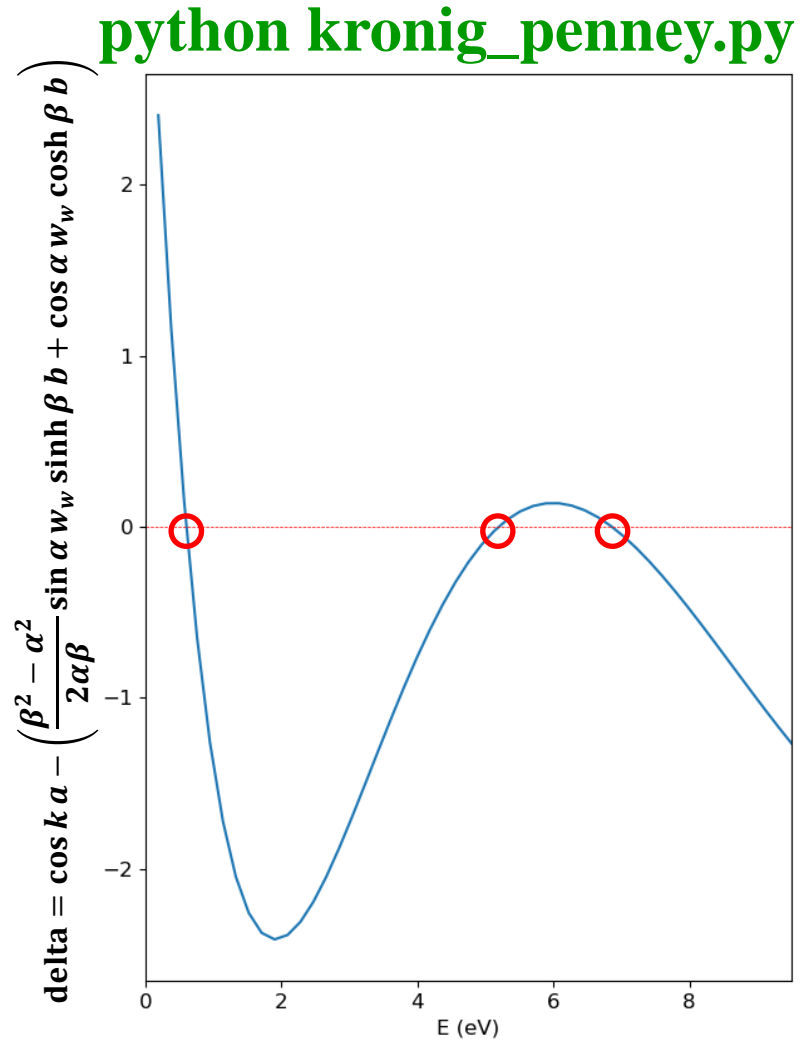
**Scan E in possible range to find all the solutions,
then use them for initial values
to obtain accurate values by Newton-Raphson method**

Program: Kronig-Penney model

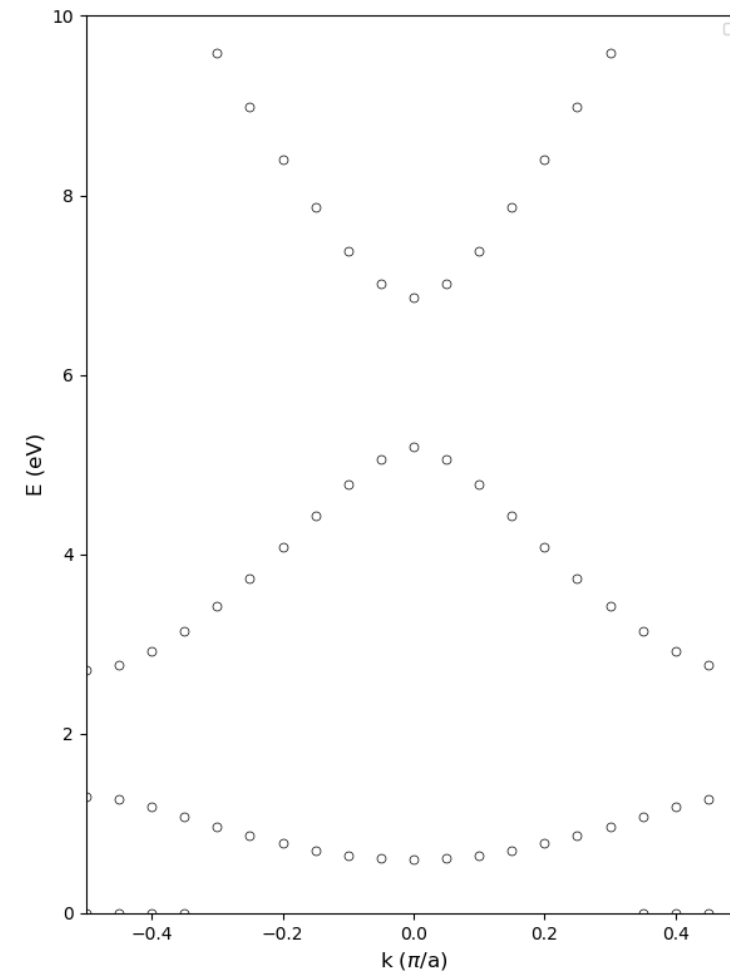
Program: **kronig_penney.py**

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

Barrier width 0.5 \AA Barrier height 10.0 eV



python kronig_penney.py band



Non-linear (NL) optimization

非線形最適化

Optimization

Objective: Find parameters \mathbf{x}_i to minimize or maximize
a objective function $F(\mathbf{x}_i)$

Maximization problem for $F(\mathbf{x}_i)$
is equivalent to minimization problem for $-F(\mathbf{x}_i)$

Examples:

- **Linear least-squares method:** A linear minimization problem for L2 norm of errors
- **Curve fitting:** A non-linear minimization problem for L2 norm of errors
- **Structure relaxation:** A non-linear minimization for total energy

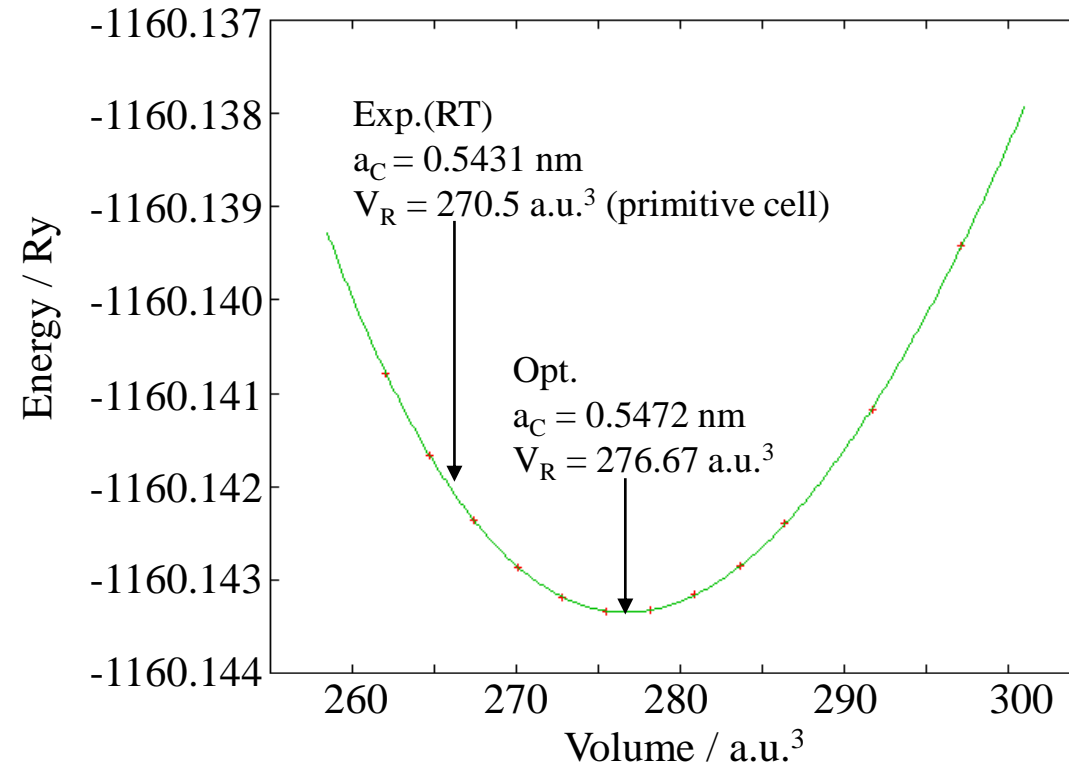
Focus on minimization problem

NL optimization of crystal structure:

Illustrative approach

安定構造: 図解による解法

Calculate total energy by quantum calculations by varying a lattice parameter
ex. Si



$$E = E_{\min} + 1/2B_0(V/V_0)^2$$

$$B_0 \text{ (GPa)} = 87.57 \text{ GPa} \quad (\text{exp: } 97.88 \text{ GPa})$$

Profile models used for spectroscopy

Lorentz function

$$I_L(x) = \frac{1}{1 + [(x - x_0)/w]^2}$$

w: half width at half maximum

Gauss function

$$I_G(x) = \frac{1}{a_w w \pi^{1/2}} \exp\left\{-\left[(x - x_0)/(a_w w)\right]^2\right\}$$
$$a_w = (\ln 2)^{-1/2} = 0.832554611$$

Voigt function:

E.g., observed is convolution of sample spectrum $I_L(x)$ and apparatus function $I_G(x)$

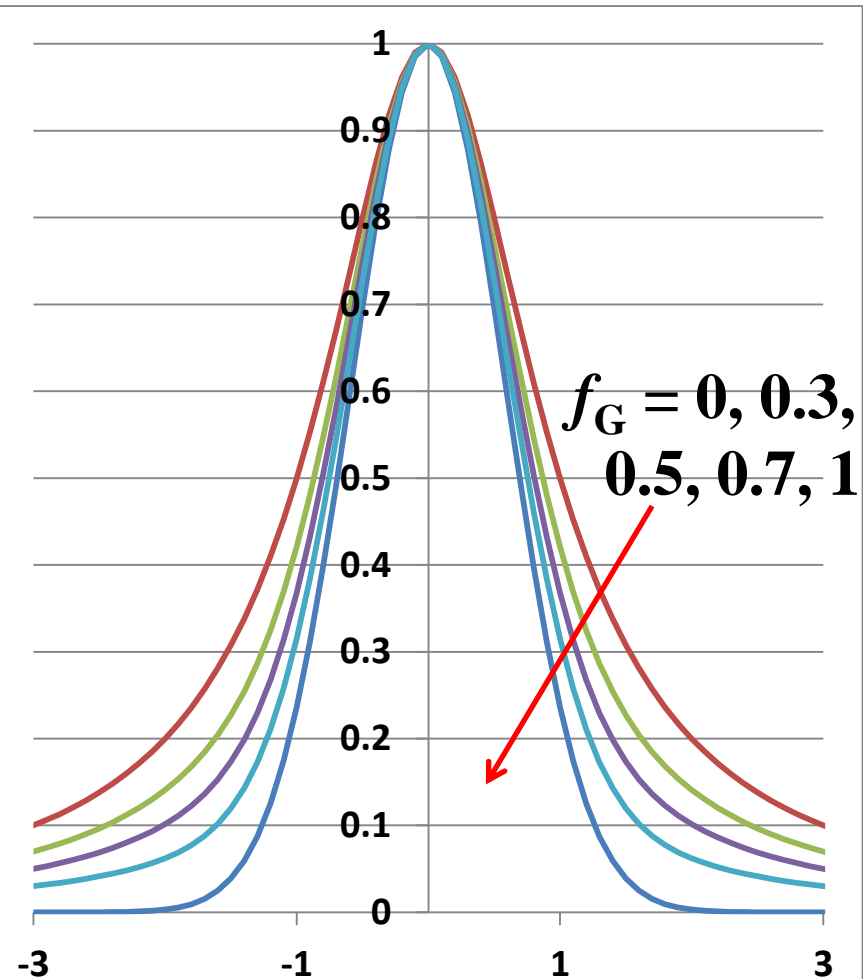
$$I_V(x) = \int_{-\infty}^{\infty} I_G(x') I_L(x - x') dx'$$
$$= \frac{a_V}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x'^2)}{a_V^2 + (x - x')^2} dx'$$

Pseudo-Voigt function:

Simplified Voigt function

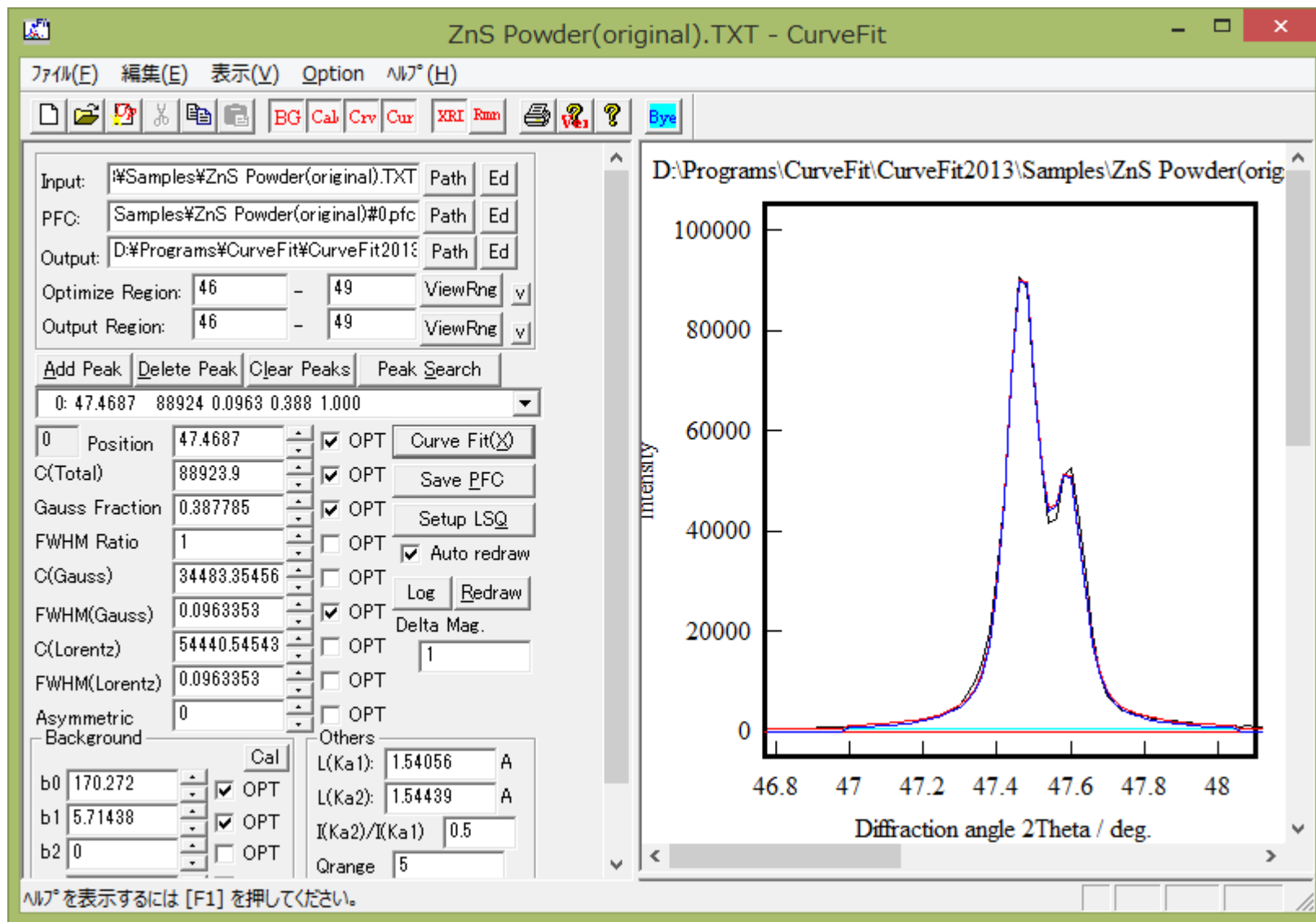
$$I_{PV}(x) = f_G I_G(x) + (1 - f_G) I_L(x)$$

f_G : Gauss fraction



Ex.: Deconvolution of powder XRD peak

Incorporate the intensity ratio from $K\alpha_1$ and $K\alpha_2$ at 2:1



Methods of non-linear (NL) optimization

To find a minimum (maximum) of **target function** $F(x)$:

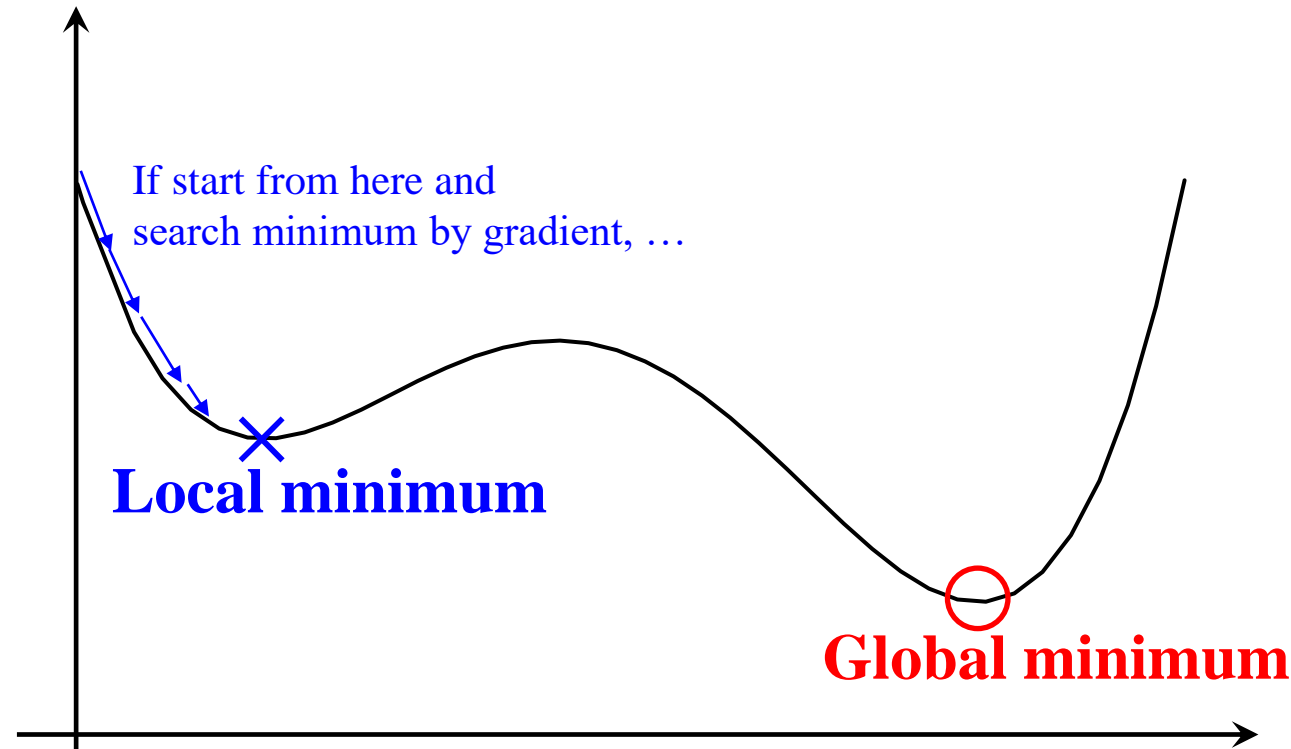
Direct search method (直接探索法)

Trial and errors to find a minimum,
but following a certain defined procedures

Gradient method (勾配法):

Use first differential to find the direction of minimum

Global minimum (大域的最小値) vs local minimum (極小値)



How to avoid to be trapped by local minimum:

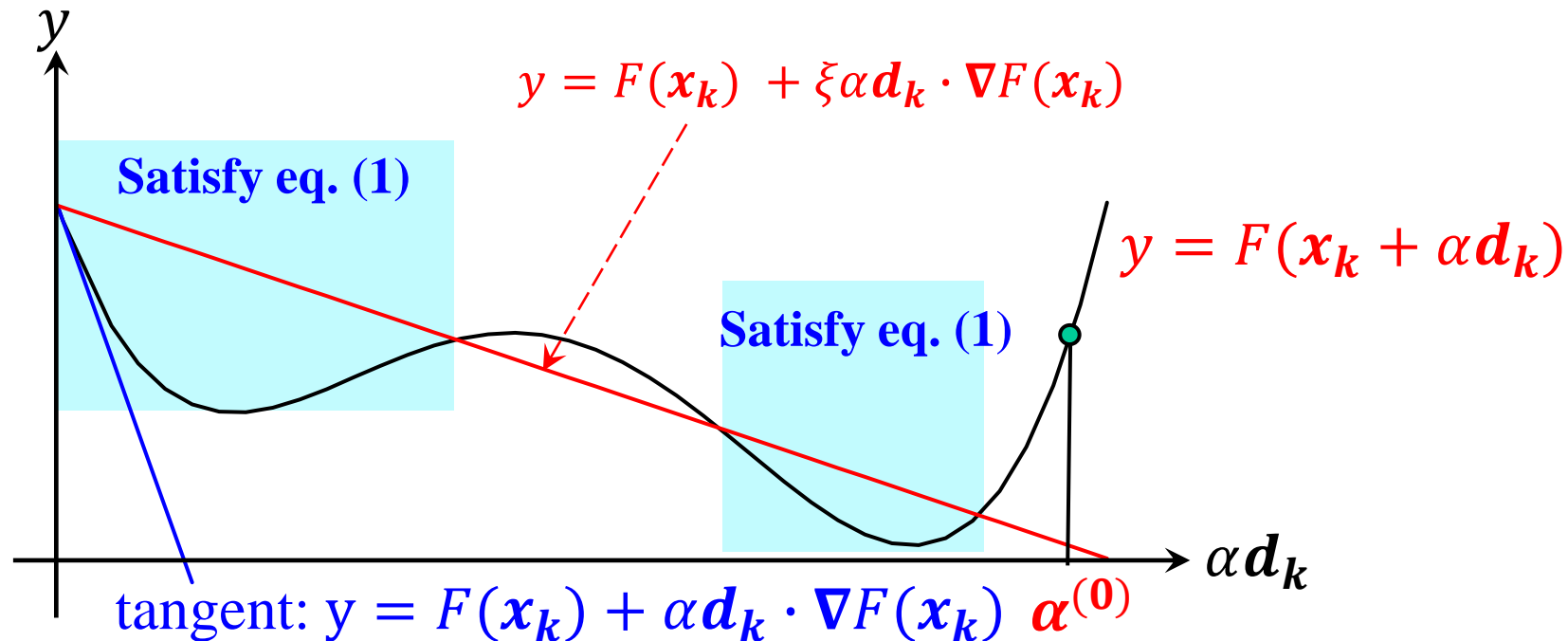
1. Employ a large initial search range
2. Not use a direct value of gradient

Line search (直線探索法): Armijo condition

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

Armijo (アルミホ) condition (eq. (1)) and algorism:

1. Provide initial \mathbf{x}_k , choose constant ξ and τ ($0 < \xi < 1$, $0 < \tau < 1$)
2. Find search direction \mathbf{d}_k (e.g., by steepest descent method)
3. Find $\alpha > 0$ so as to satisfy $F(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq F(\mathbf{x}_k) + \xi \alpha \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k)$ (1)
 - (i) $\beta_{k,0} = 1, i = 0$
 - (ii) if $F(\mathbf{x}_k + \beta_{k,i} \mathbf{d}_k) \leq F(\mathbf{x}_k) + \xi \beta_{k,i} \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k)$ go to step 4, or go to (iii)
 - (iii) $\beta_{k,i+1} = \tau \beta_{k,i}$ and go to (ii)
4. $\alpha = \beta_{k,i}$

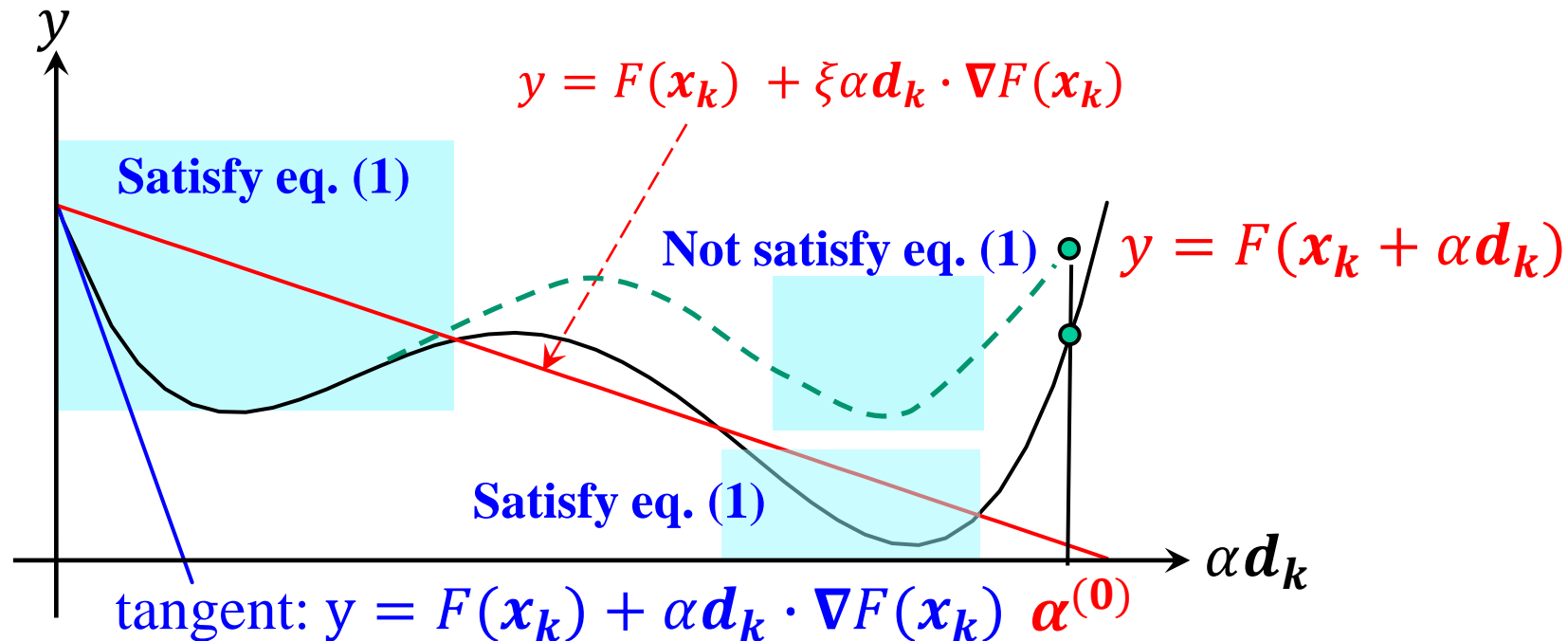


Line search (直線探索法): Armijo condition

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

Armijo (アルミホ) condition (eq. (1)) and algorism:

1. Provide initial \mathbf{x}_k , choose constant ξ and τ ($0 < \xi < 1$, $0 < \tau < 1$)
2. Find search direction \mathbf{d}_k (e.g., by steepest descent method)
3. Find $\alpha > 0$ so as to satisfy $F(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq F(\mathbf{x}_k) + \xi \alpha \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k)$ (1)
 - (i) $\beta_{k,0} = 1, i = 0$
 - (ii) if $F(\mathbf{x}_k + \beta_{k,i} \mathbf{d}_k) \leq F(\mathbf{x}_k) + \xi \beta_{k,i} \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k)$ go to step 4, or go to (iii)
 - (iii) $\beta_{k,i+1} = \tau \beta_{k,i}$ and go to (ii)
4. $\alpha = \beta_{k,i}$



Line search (直線探索法): Wolfe condition

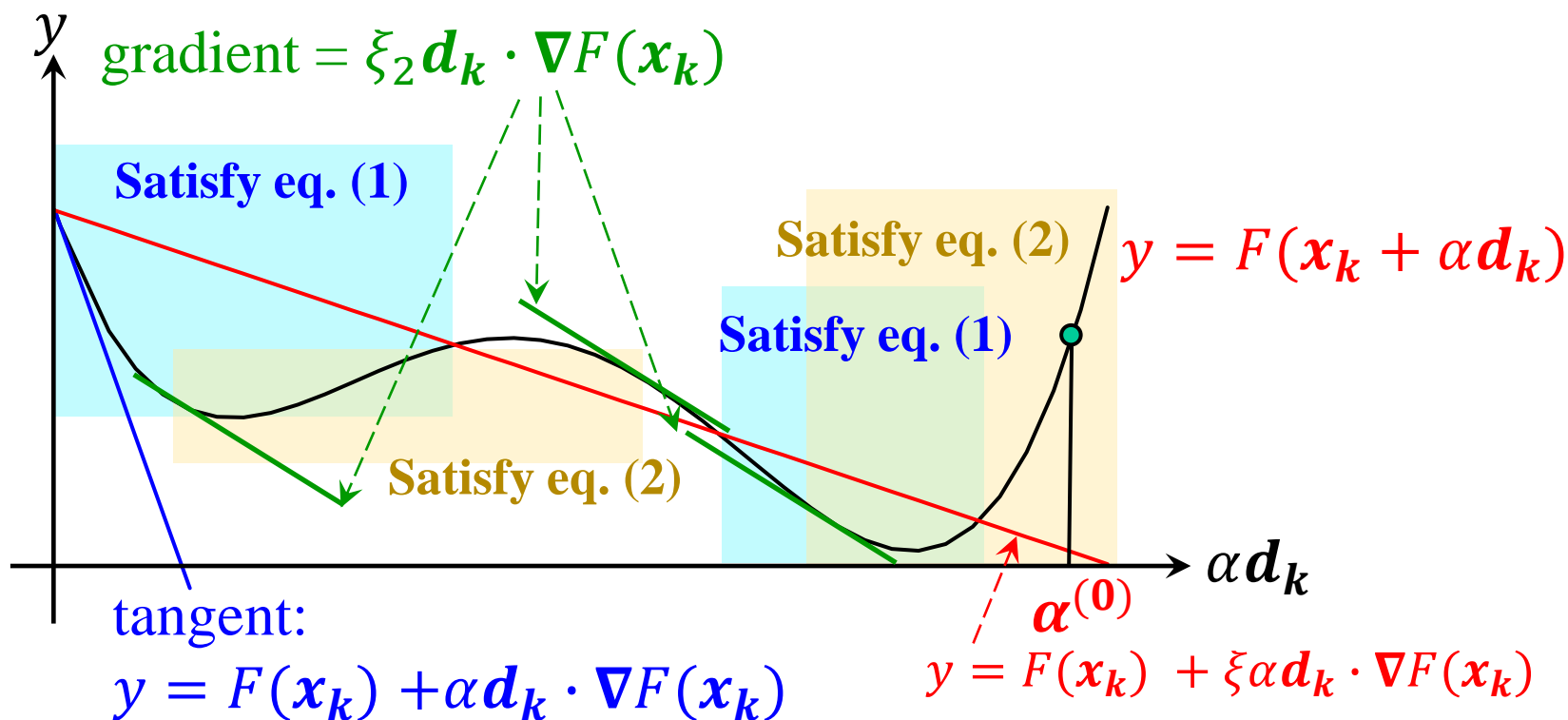
矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

Wolfe (ウルフ) condition:

1. Find search direction \mathbf{d}_k (e.g., by steepest descent method)
2. Choose constants ξ_1 and ξ_2 that satisfy $0 < \xi_1 < \xi_2 < 1$
3. Find $\alpha > 0$ so as to satisfy:

$$F(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq F(\mathbf{x}_k) + \xi \alpha \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k) \quad (1)$$

$$\xi_2 \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k) \leq \mathbf{d}_k \cdot \nabla F(\mathbf{x}_k + \alpha \mathbf{d}_k) \quad (2)$$



Bisection method (二分法) vs Golden-section search (黄金分割探索)

Bisection method: Find solution of $f(x) = 0$ for monotonous continuous function

Unique solution exists in the range $[x_0^{(0)}, x_2^{(0)}]$ if $f(x_0^{(0)})f(x_2^{(0)}) < 0$

Add $x_1^{(0)}$ in $[x_0^{(0)}, x_2^{(0)}]$ ($x_0^{(0)} < x_1^{(0)} < x_2^{(0)}$)

Case 1: If $f(x_0^{(0)})f(x_1^{(0)}) < 0$, solution is in $[x_0^{(0)}, x_1^{(0)}]$

Next search range is reduced to $x_0^{(1)} := x_0^{(0)}$, $x_1^{(1)} := x_3^{(0)} = \frac{x_0^{(0)} + x_1^{(0)}}{2}$, $x_2^{(1)} := x_1^{(0)}$

Case 2: If $f(x_1^{(0)})f(x_2^{(0)}) < 0$, solution is in $[x_1^{(0)}, x_2^{(0)}]$

Next search range is reduced to: $x_0^{(1)} := x_1^{(0)}$, $x_1^{(1)} := x_3^{(0)} = \frac{x_1^{(0)} + x_2^{(0)}}{2}$, $x_2^{(1)} := x_2^{(0)}$

Golden-section search: Find minimum for single downward convex continuous func $f(x)$

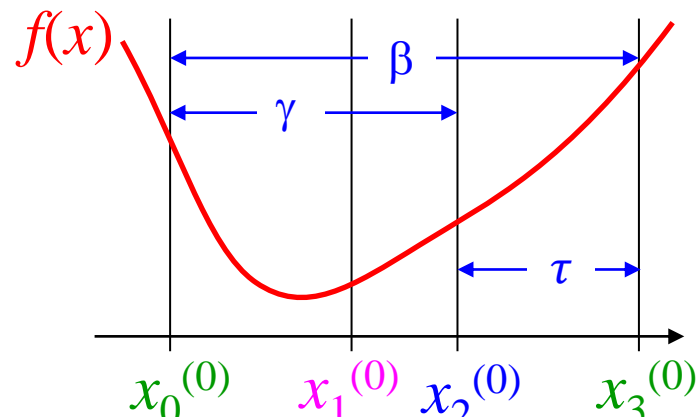
Unique solution exists in the range $[x_0^{(0)}, x_3^{(0)}]$ if $f(x_1^{(0)}) < f(x_0^{(0)}), f(x_3^{(0)})$ for $x_0^{(0)} < x_1^{(0)} < x_3^{(0)}$

Add $x_2^{(0)}$ in $[x_0^{(0)}, x_3^{(0)}]$ ($x_0^{(0)} < x_1^{(0)} < x_2^{(0)} < x_3^{(0)}$)

Case 1: if $f(x_1^{(0)}) < f(x_2^{(0)})$, solution is in $[x_0^{(0)}, x_2^{(0)}]$

Replace $x_2^{(0)}$ with $x_4^{(0)}$ in $[x_0^{(0)}, x_1^{(0)}]$

Next search range is reduced to $x_0^{(1)} := x_0^{(0)} < x_1^{(1)} := x_4^{(0)} < x_2^{(1)} := x_1^{(0)} < x_3^{(1)} := x_2^{(0)}$



Golden-section search (黄金分割探索)

For downward convex continuous function, unique solution exists

in the range $[x_0^{(0)}, x_3^{(0)}]$ if $f(x_1^{(0)}), f(x_2^{(0)}) < f(x_0^{(0)}), f(x_3^{(0)})$ for $x_0^{(0)} < x_1^{(0)} < x_2^{(0)} < x_3^{(0)}$

Case 1: if $f(x_1^{(0)}) < f(x_2^{(0)})$, solution is in $[x_0^{(0)}, x_2^{(0)}]$

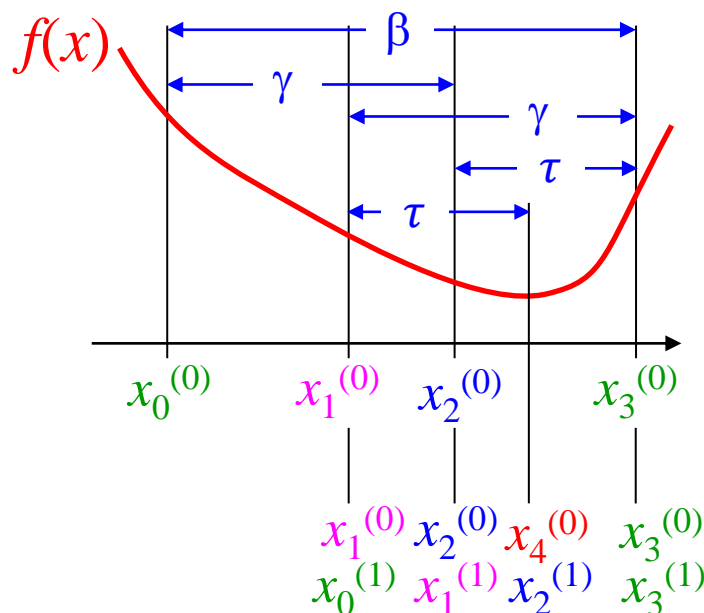
Replace $x_2^{(0)}$ with $x_4^{(0)}$ in $[x_0^{(0)}, x_1^{(0)}]$

Next search range is reduced to $x_0^{(1)} := x_0^{(0)} < x_1^{(1)} := x_4^{(0)} < x_2^{(1)} := x_1^{(0)} < x_3^{(1)} := x_2^{(0)}$

Case 2: if $f(x_1^{(0)}) > f(x_2^{(0)})$, solution is in $[x_1^{(0)}, x_3^{(0)}]$

Replace $x_0^{(0)}$ with $x_4^{(0)}$ in $[x_2^{(0)}, x_3^{(0)}]$

Next search range is reduced to $x_0^{(1)} := x_1^{(0)} < x_1^{(1)} := x_2^{(0)} < x_2^{(1)} := x_4^{(0)} < x_3^{(1)} := x_3^{(0)}$



Strategy: keep the ratio of $x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, x_3^{(k)}$ constant for iteration steps

$$\beta = x_3^{(k)} - x_0^{(k)}$$

$$\gamma = x_2^{(k)} - x_0^{(k)} = x_3^{(k)} - x_1^{(k)}$$

$$\tau = \beta - \gamma$$

$$x_4^{(k)} = x_1^{(k)} + \tau$$

To keep the ratio for next step (k+1)

$$\beta : \gamma = x_3^{(k+1)} - x_0^{(k+1)} : x_2^{(k+1)} - x_0^{(k+1)}$$

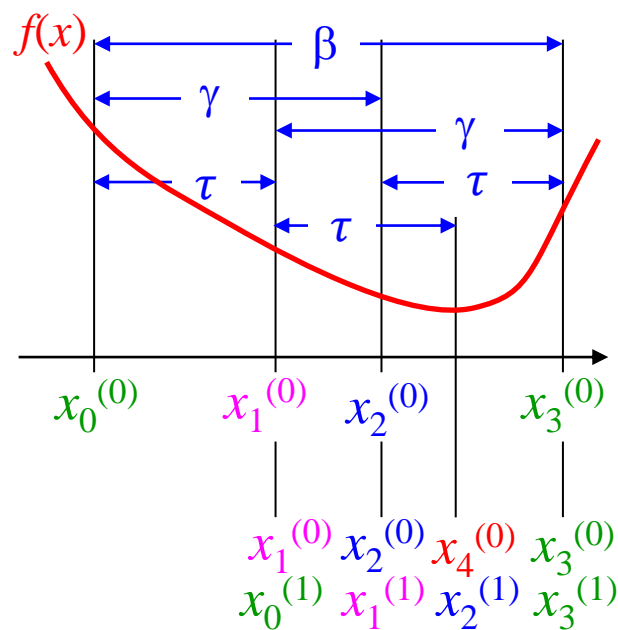
$$= x_3^{(k)} - x_1^{(k)} : x_4^{(k)} - x_1^{(k)} = \gamma : \tau$$

$$\tau = \beta - \gamma \text{ から}$$

$$\frac{\beta}{\gamma} = \frac{1+\sqrt{5}}{2} \text{ Golden number}$$

Golden-section search (黄金分割探索)

Minimum solution of downward convex continuous function $f(x)$



$$\frac{\beta}{\gamma} = \frac{1+\sqrt{5}}{2}$$

$$\eta = \frac{\tau}{\beta} = \frac{\beta-\gamma}{\beta} = 1 - \frac{2}{\sqrt{5}+1} = \frac{\sqrt{5}-1}{\sqrt{5}+1}$$

1. For $x_0^{(0)} < x_1^{(0)} < x_2^{(0)} < x_3^{(0)}$, assign initial parameters as:

$$\beta^{(0)} = x_3^{(0)} - x_0^{(0)}$$

$$\tau^{(0)} = \eta \beta^{(0)}$$

$$x_1^{(0)} = x_0^{(0)} + \tau^{(0)}$$

$$x_2^{(0)} = x_3^{(0)} - \tau^{(0)}$$

2. Terminate if $|\beta^{(k)}| < \text{eps}$

$$3. \quad \beta^{(k+1)} = \tau^{(k)}$$

$$\tau^{(k+1)} = \eta \beta^{(k+1)}$$

4. If $f(x_1^{(k)}) < f(x_2^{(k)})$, substitute $x_3^{(k)}$ for $x_4^{(k)} = x_2^{(k)} - \tau^{(k)}$ as:

$$x_0^{(k+1)} = x_0^{(k)}, x_1^{(k+1)} = x_2^{(k)} - \tau^{(k)}, x_2^{(k+1)} = x_1^{(k)}, x_3^{(k+1)} = x_2^{(k)}$$

If $f(x_1^{(k)}) > f(x_2^{(k)})$, substitute $x_0^{(k)}$ for $x_4^{(k)} = x_2^{(k)} + \tau^{(k)}$ as:

$$x_0^{(k+1)} = x_1^{(k)}, x_1^{(k+1)} = x_2^{(k)}, x_2^{(k+1)} = x_1^{(k)} + \tau^{(k)}, x_3^{(k+1)} = x_3^{(k)}$$

Go to step 1

Methods of non-linear (NL) optimization

To find a minimum (maximum) of **target function** $F(x)$:

Direct search method (直接探索法)

Trial and errors to find a minimum,
but following a certain defined procedures

Gradient method (勾配法):

Use first differential to find the direction of minimum

Steepest descent method (SD, 最急降下法)

Search minimum/maximum only by first derivatives

Objective function (multi-variable, x_j): $F(x_j)$

Concept: Minimum/Maximum may be found in the direction $(\partial F(x_j)/\partial x_i)$

$$x_i^{(k+1)} = x_i^{(k)} - \alpha \partial F(x_j^{(k)}) / \partial x_i$$

Need to choose/find an appropriate α so as to take the minimum $F(x_j^{(k)})$

Variations to choose α :

(i) Simple: Choose small α

(ii) Direct search (直接探索)

Armijo / Wolfe condition

Steepest Descend (SD) method (最急降下法)

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

Search minimum only by first derivatives. Simplest one among gradient methods

- **SD:** S^2 would decrease in the vector $-(df / dx_i)dx_i$

$$x_i^{(k+1)} = x_i^{(k)} - \alpha(df / dx_i)$$

α_k may be a small constant step

or determined by a line search method

ex. in right figure:

$$S^2 = f(x_i) = 5x_1^2 + x_2^2, \text{ initial } x_1 = 0.7, x_2 = 1.5$$

- **SD method**

$\alpha = 0.3$: Diverged (not shown in the graph)

0.2, 0.15: Converged, but oscillated

0.1: Reach final solution by one cycle calculation

0.01: Not oscillated, but slowly converged

- **Newton method**

One cycle calculation provides the final solution

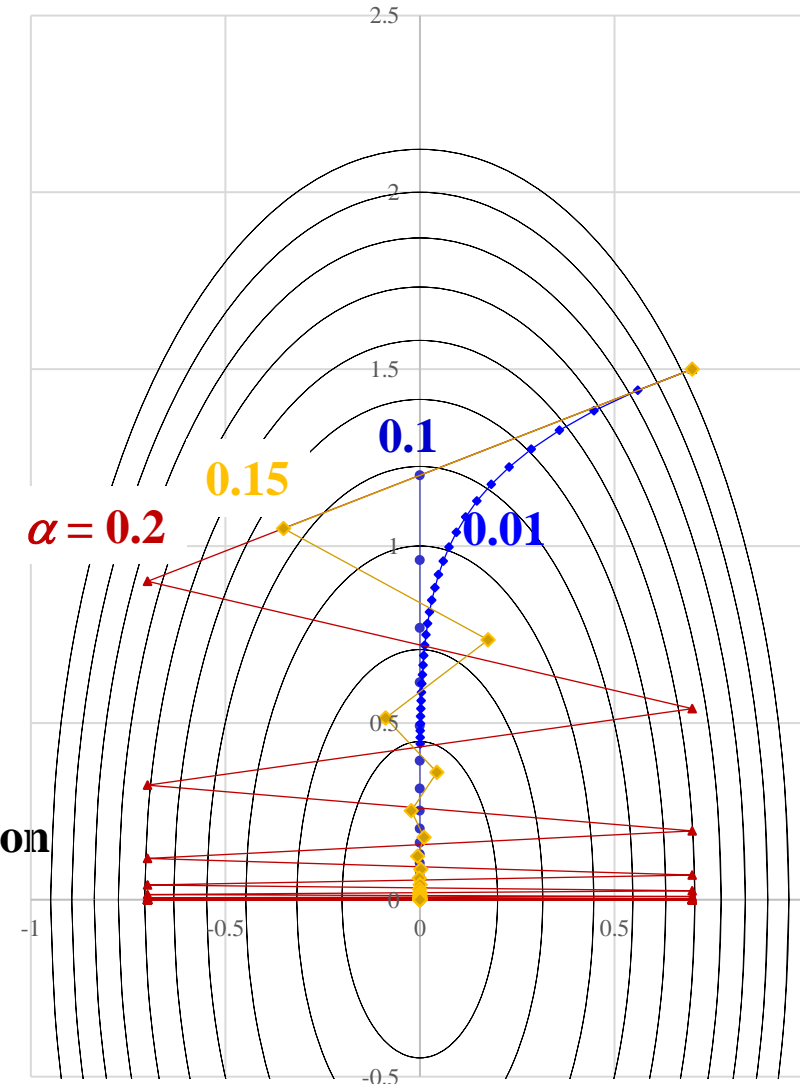
for quadratic problems

楕円問題の場合は一度目の計算で最適値に到達

Problem: If S^2 is highly anisotropic, the SD direction would be different largely from the minimization

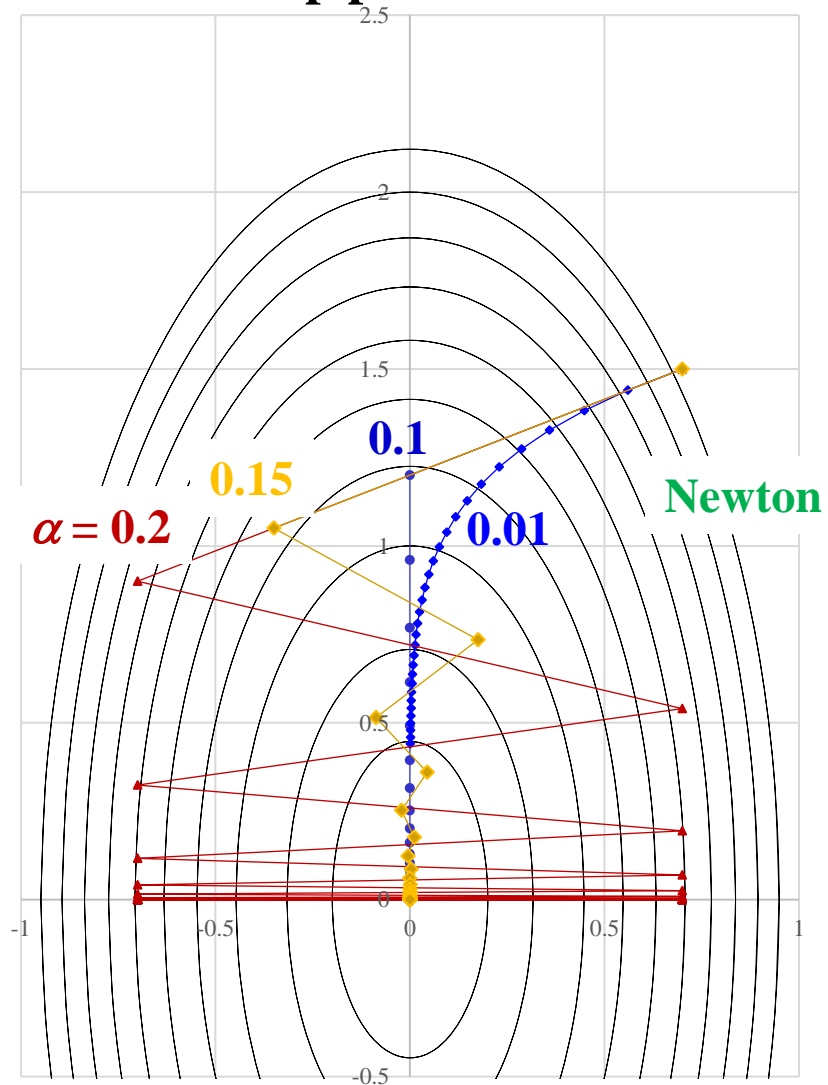
direction S^2 が大きく非対称な場合、最急勾配方向は最小値方向とは大きく異なることがある

=> **Conjugate Gradient (CG) method** (共役勾配法)

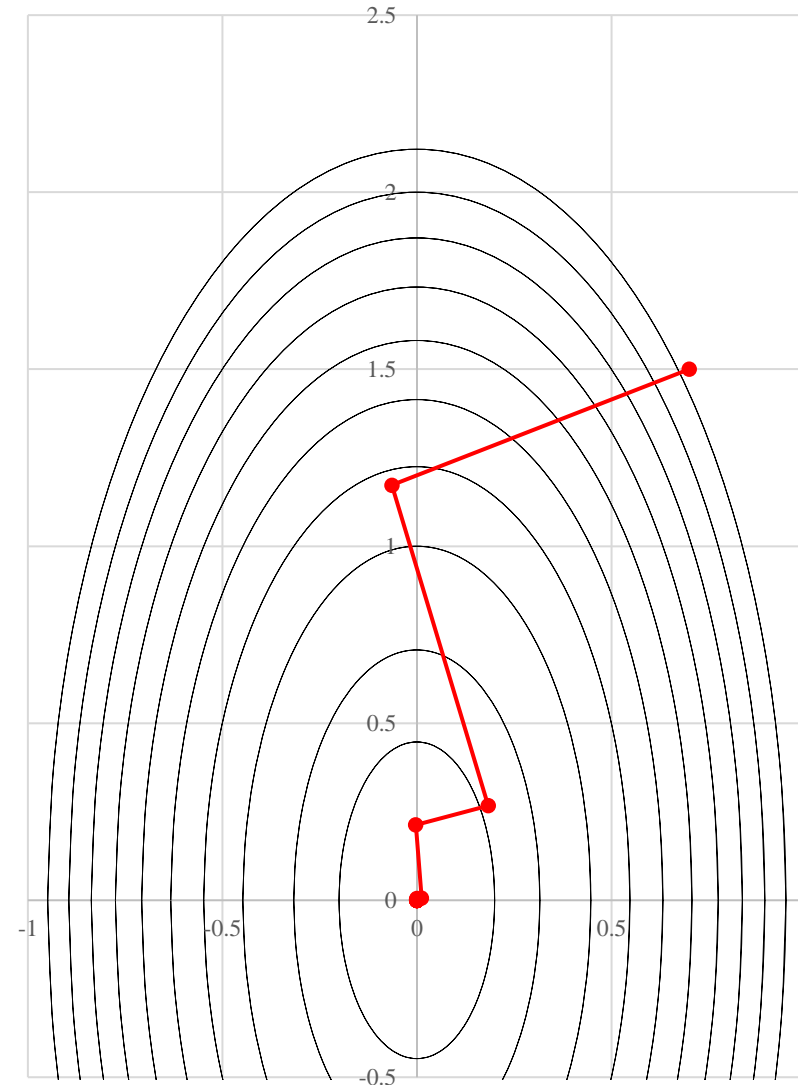


Steepest descend method

Without direct search:
use fixed step parameter



With direct search



SD method in Deep Learning

- All batch data are divided to mini batches, and apply SD to each mini batch

Example:

$$S^2 = f(a, b; x_1, x_2) = ax_1^2 + bx_2^2, \quad a = 5, b = 1$$

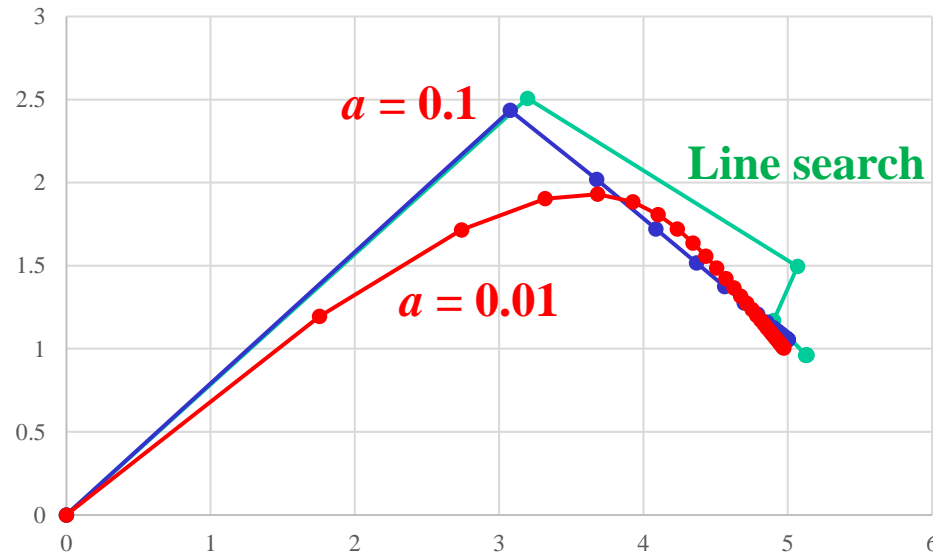
1000 batch data $(x_{1i}, x_{2i}, f(x_i))$ are generated at random

(note: the data were re-generated for different runs)

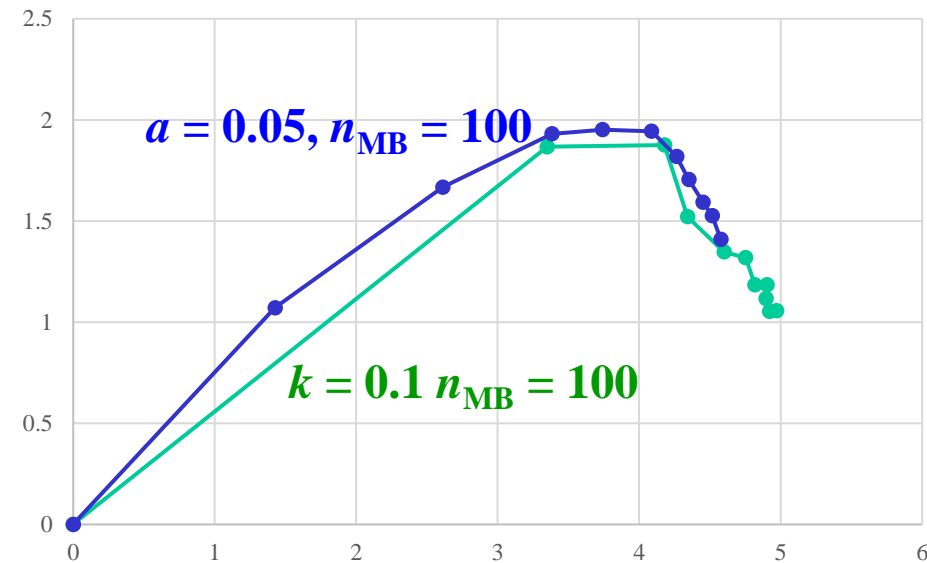
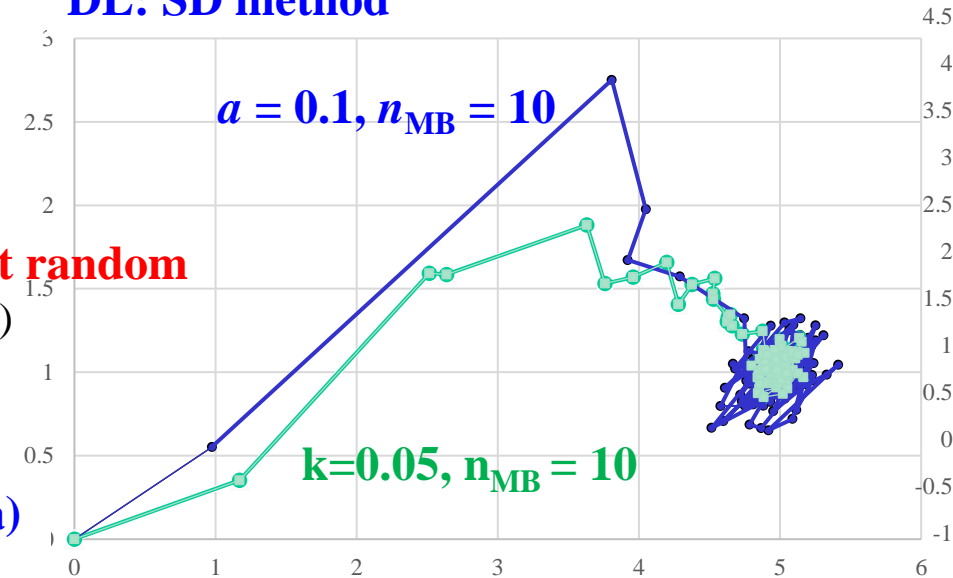
Speculate a and b

Initial $a = 0, b = 0$

SD (Convergence iterations with all batch data)



DL: SD method



Multiple variable Newton-Raphson method

Extend to **multiple variable optimization**: Minimize $F(x_j)$

$$f_k(x_j) = \partial F(x_j) / \partial x_k = 0$$

To solve $f_k(x_j) = 0$ ($k, j = 1, 2, \dots, N$)

$$f_k(x_j + \delta x_j) \sim f_k(x_j) + \sum_{k'} \delta x_{k'} \partial f_k(x_j) / \partial x_{k'} = 0$$

$$\Rightarrow x_{j,1} = x_{j,0} - (\partial f_k(x_j) / \partial x_{k'})^{-1} (f_k) = x_{l,0} - (F''_{kk'})^{-1} (F'_k)$$

$$F''_{kk'} = \frac{\partial^2 F(x_j)}{\partial x_k \partial x_{k'}} \quad \text{Hessian matrix (ヘッセ行列)}$$

(ヘッセ行列の固有値をヘッシアンと呼ぶ)

Hessian matrix is not always positive definite (正定値であるとは限らない)

(Maximum, Saddle point 極大値、鞍点)

$\Rightarrow F''$ does not always give decreasing direction

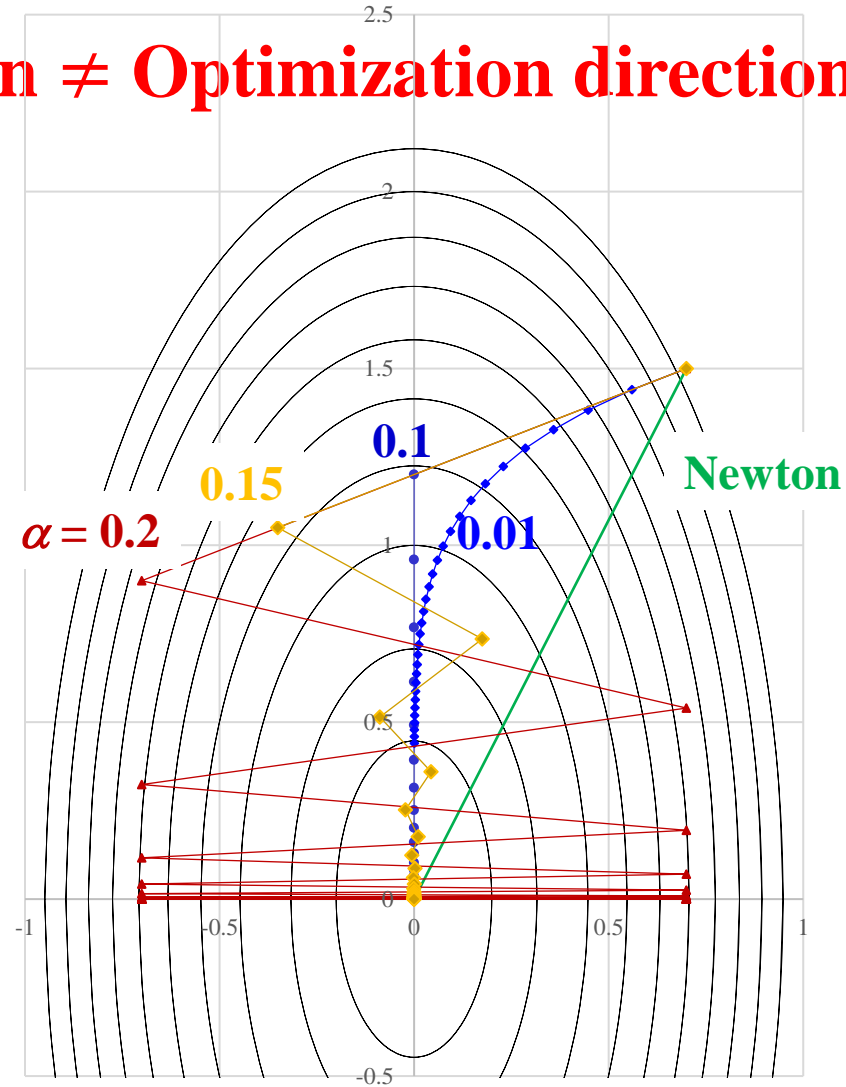
Convert F'' to positive definite and suppress divergence

$$x_{l,1} = x_{l,0} - (F''_{kk'} + \lambda I)^{-1} (F'_k)$$

λ : **Dumping Factor**

SD vs. Newton-Raphson methods

Steepest direction \neq Optimization direction

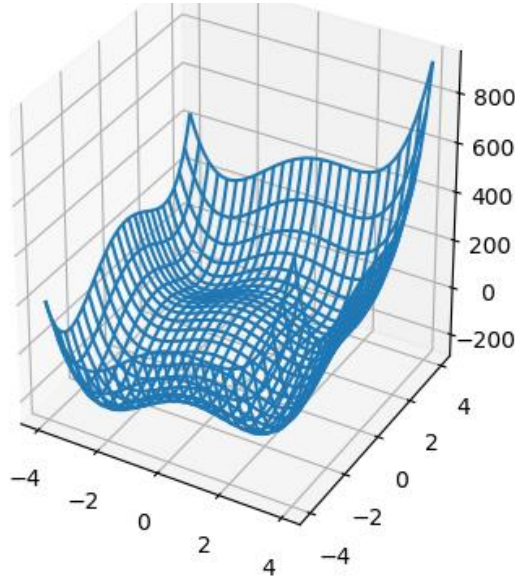


Improve SD method to follow optimization directions

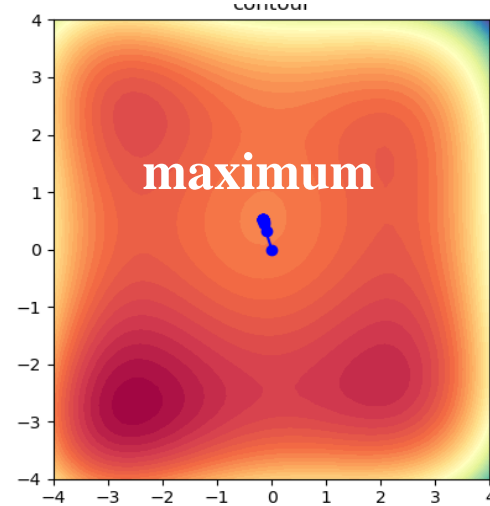
Program: optimize-newton-raphson2d.py

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

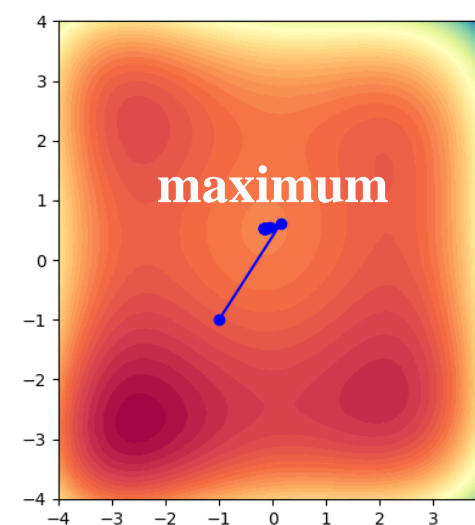
Usage: `python optimize-newton-raphson2d.py x0 y0`



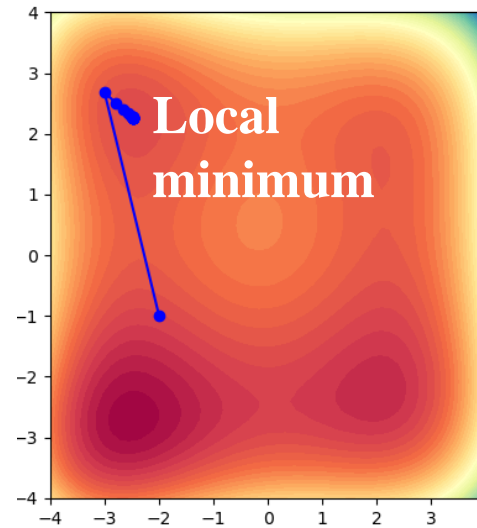
From (0.0 0.0) Newton



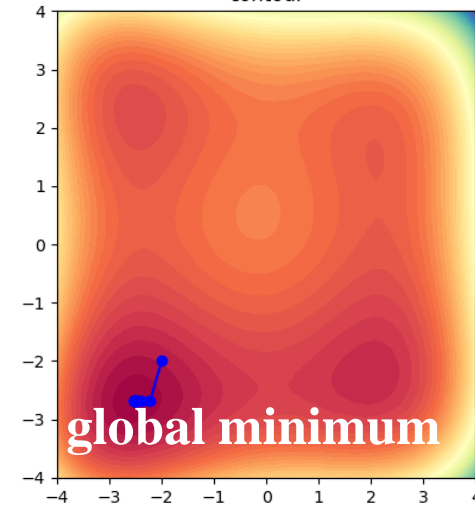
From (-1.0 -1.0)



From (-2.0 -1.0)



From (-2.0 -2.0)



Quasi-Newton method (準Newton法)

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

Target function to minimize: $F(x_j)$

Iteration: $x_j^{(i+1)} = x_j^{(i)} - (\partial^2 F / \partial x_k \partial x_{k'})^{-1} (\partial F / \partial x_k)$

$F''_{kk'} = \partial^2 F / \partial x_k \partial x_{k'}$: Hessian (ヘッセ) matrix

Issues of Newton method:

- (1) Calculation of Hessian matrix is very high cost as it is a 2D matrix
- (2) Eigen value of Hessian matrix can be negative => lead to maximum
- (3) Easy to diverge

Quasi-Newton method:

- (1,2) Hessian matrix is approximated from 1st differentials
- (3) Line search algorithm is applied along the search direction
 $-(\partial^2 F / \partial x_k \partial x_{k'})^{-1} (\partial F / \partial x_k)$

Davidon-Fletcher-Powell (DFP) method

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

$$F(x_j^{(k)} + \alpha d) = F(x_j^{(k)}) + \alpha \nabla F(x_j^{(k)})^T d + \frac{1}{2} \alpha^2 d^T B^{(k)} d \sim 0$$

Search direction d is determined from $B^{(k)} d = -\nabla F(x_j^{(k)})$

DFP method: The first formulation of quasi-Newton method

$$s^{(k)} = x^{(k+1)} - x^{(k)}, \quad y^{(k)} = \nabla F(x_j^{(k+1)}) - \nabla F(x_j^{(k)})$$

$$\begin{aligned} B^{(k+1)} &= B^{(k)} + \frac{(y^{(k)} - B^{(k)} s^{(k)}) \cdot y^{(k)T} + y^{(k)} \cdot (y^{(k)} - B^{(k)} s^{(k)})^T}{s^{(k)T} \cdot y^{(k)}} \\ &\quad - \frac{s^{(k)T} \cdot (y^{(k)} - B^{(k)} s^{(k)})}{(s^{(k)T} \cdot y^{(k)})^2} y^{(k)} \cdot y^{(k)T} \\ &= B^{(k)} - \frac{B^{(k)} s^{(k)} \cdot y^{(k)T} + y^{(k)} \cdot (B^{(k)} s^{(k)})^T}{s^{(k)T} \cdot y^{(k)}} + \left(1 + \frac{s^{(k)T} B^{(k)} s^{(k)}}{s^{(k)T} \cdot y^{(k)}} \right) \end{aligned}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) method

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

BFGS method: Regarded as most efficient among quasi-Newton methods

$$s^{(k)} = x^{(k+1)} - x^{(k)}, \quad y^{(k)} = \nabla F(x_j^{(k+1)}) - \nabla F(x_j^{(k)})$$

$$B^{(k+1)} = B^{(k)} - \frac{B^{(k)} s^{(k)} (B^{(k)} s^{(k)})^T}{s^{(k)T} B^{(k)} s^{(k)}} + \frac{y^{(k)} y^{(k)T}}{s^{(k)T} \cdot y^{(k)}}$$

Algorithm:

STEP 0: Provide initial values $x^{(0)}$ and initial matrix $B^{(0)}$ (can be unit matrix)

STEP 1: Search direction $d^{(k)}$ is determined from $B^{(k)} d = -\nabla F(x_l^{(k)})$

STEP 2: Step width $\alpha^{(k)}$ is determined by **line search algorism**

STEP 3: Calculate $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$

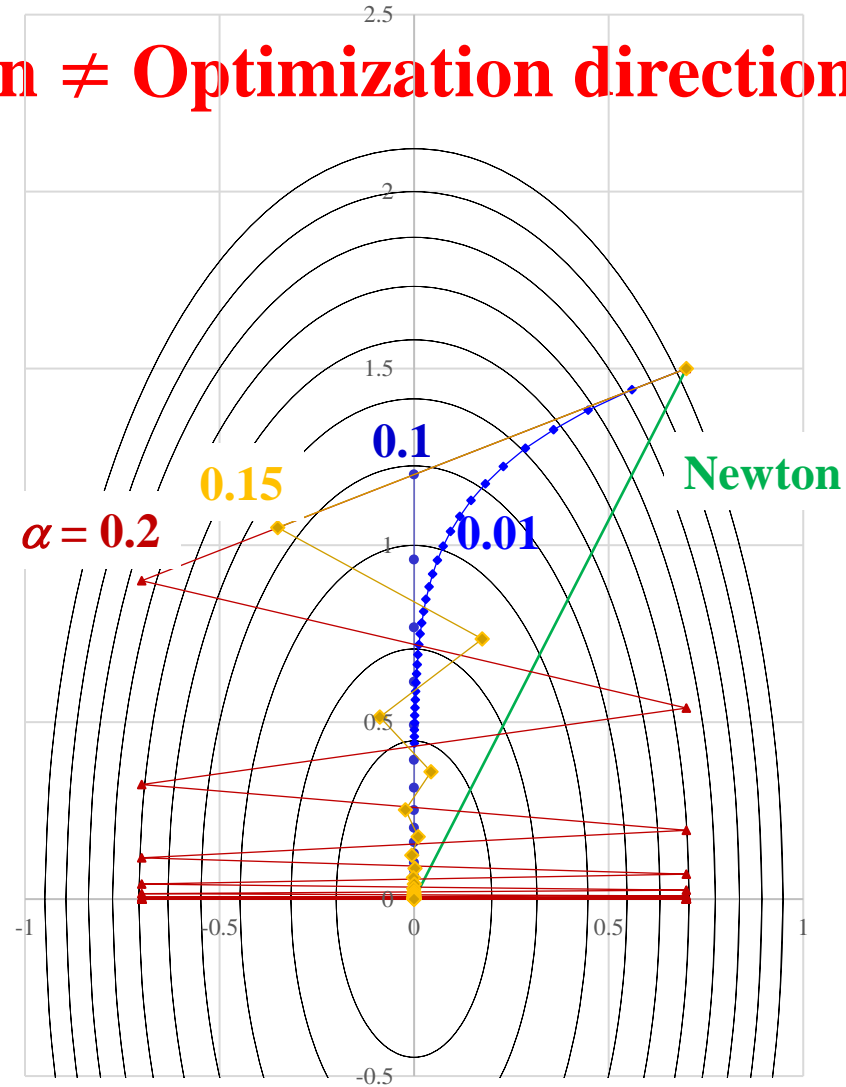
STEP 4: End if self-consistency is achieve.

If not, go to STEP 5

STEP 5: Calculated $s^{(k)}$ and $y^{(k)}$, and then $B^{(k+1)}$, and go to STEP 1

SD vs. Newton-Raphson methods

Steepest direction \neq Optimization direction



Improve SD method to follow optimization directions

APPENDIX

related to linear least-squares methods

最尤推定法

尤度関数とは:

事象 (x_k) が起こる確率を、既知のパラメータ (a_k) の確率密度関数

$$P(X = x_i | a_k) = \prod_i \left\{ \frac{1}{\sqrt{2\pi}\sigma_i} \exp \left[-\frac{\varepsilon_i(a_k)^2}{2\sigma_i^2} \right] \right\} = \prod_i \left(\frac{1}{\sqrt{2\pi}\sigma_i} \right) \cdot \exp \left[-\sum_i \frac{\varepsilon_i(a_k)^2}{2\sigma_i^2} \right]$$

などとする、逆に $X = (x_i)$ がわかっていると、

パラメータ (a_k) がどれだけ尤もらしいか (尤度) を表す確率密度関数とみなし、

上記の確率密度関数を変数 (a_k) の関数として

尤度関数 $P(a_i) = P(x_i | a_i)$ という。

最尤推定法

誤差 $\varepsilon_i = f(x_i, a_i) - y_i$ が分散 σ_i の正規分布に従うとする。データ (x_i, y_i) に対するパラメータ (a_i) の尤度関数は

$$P(a_i) = \prod_i \left(\frac{1}{\sqrt{2\pi}\sigma_i} \right) \cdot \exp \left[-\sum_i \frac{\varepsilon_i^2}{2\sigma_i^2} \right]$$

尤度を最大化するパラメータを求めるのが「最尤推定法」。

$$\max P(a_i) = \max \ln P(a_i) = \min \sum_i \frac{\varepsilon_i^2}{\sigma_i^2}: \text{最小二乗法に一致する}$$

FET特性の解析: 飽和領域

$$I_{DS} = \frac{W}{L} \mu C_{OX} \left[(V_{GS} - V_{th}) V_{DS} - \frac{V_{DS}^2}{2} \right]$$

$$V_{DS} > V_p = V_{GS} - V_{th}$$

$$I_{DS} = \frac{W}{2L} \mu C_{OX} (V_{GS} - V_{th})^2$$

$$I_{DS}^{1/2} = \sqrt{\frac{W}{2L} \mu C_{OX} (V_{GS} - V_{th})^2}$$

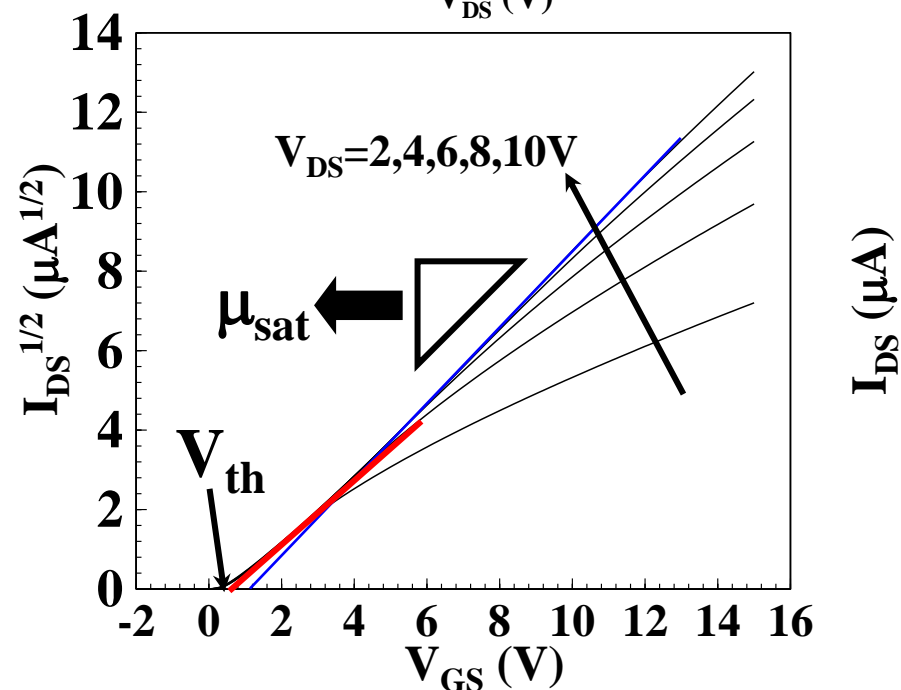
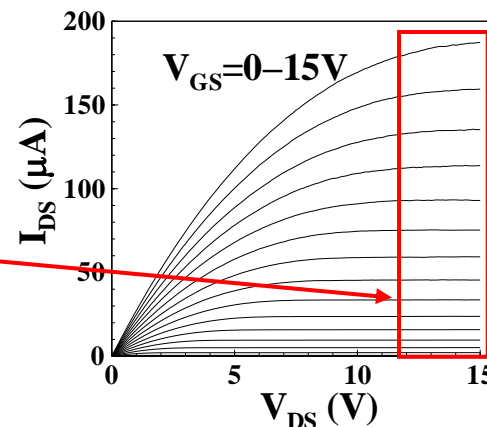
$I_{DS}^{1/2}$ vs. V_{GS} をプロット

V_{GS} 軸切片: V_{th}

傾き: 飽和移動度

Saturation mobility, μ_{sat}

a-IGZO TFT



ゲート容量 C_{OX} の計算

ゲート絶縁体: アモルファス SiO_2

誘電率: $11.9\epsilon_0$ ($\epsilon_0 = 8.854418782\text{e-}12 \text{ C}^2\text{N}^{-1}\text{m}^{-2}$)

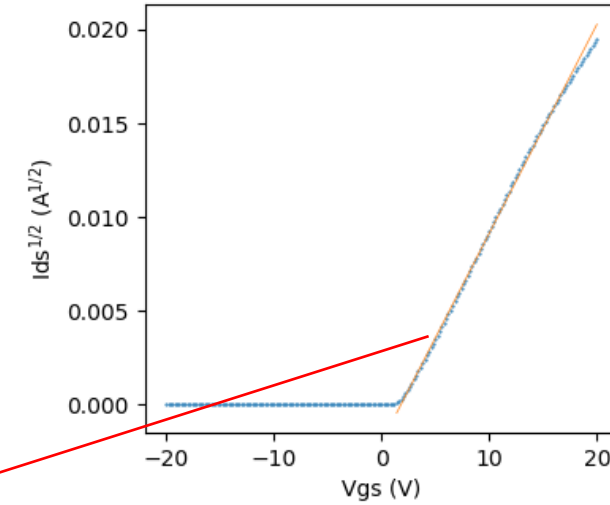
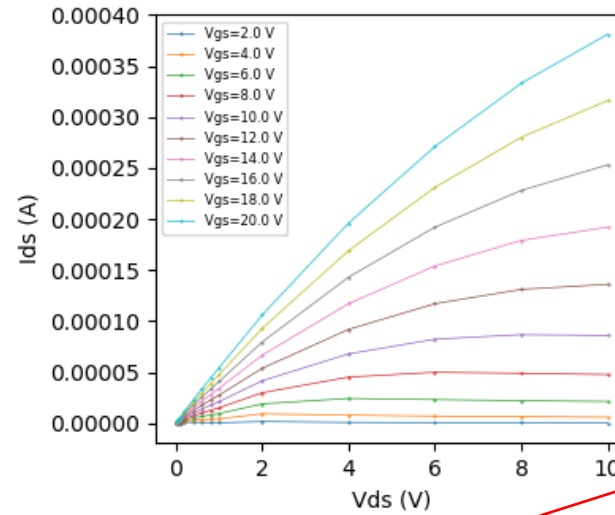
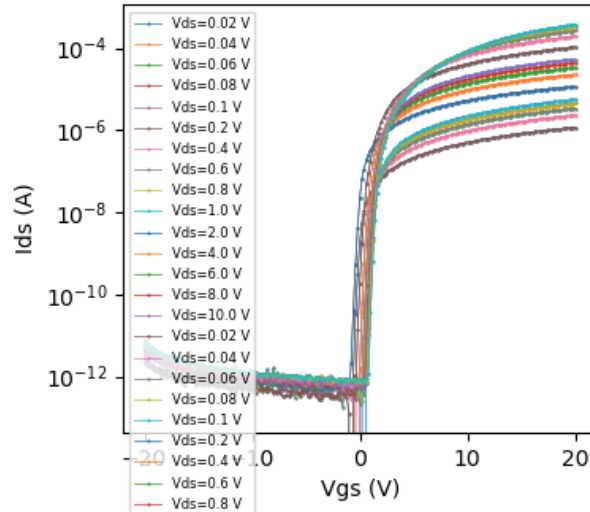
厚さ 100 nm

1 m² 当たり静電容量

$$C_{OX} = 11.9\epsilon_0 / 100\text{e-}9 [\text{m}] = 0.00105 \text{ F/m}^2$$

プログラム

TFTiv.py



直線に見える 1.9 ~ 10.0 V で、numpy.polyfit() で一次多項式にフィッティング

```
ai = np.polyfit(xfit, yfit, 1)
```

```
# y = ai[1] + ai[0]x
```

```
Vth = -ai[1] / ai[0] = 1.794 V
```

```
 $\frac{dI_{gs}^{1/2}}{dV_{gs}} = ai[0] = 0.001114 \text{ A}^{1/2}/\text{V}$ 
```

$$I_{DS}^{1/2} = \sqrt{\frac{W}{2L} \mu C_{OX} (V_{GS} - V_{th})}$$

$$\mu_{sat} = \frac{(dI_{gs}^{1/2}/dV_{gs})^2}{\frac{W}{2L} C_{OX}} = 0.000392 \text{ m}^2/\text{Vs} = 3.92 \text{ cm}^2/\text{Vs}$$

プログラム (抜粋)

TFTiv.py

```
import re    # 正規表現モジュールを読み込む
```

```
#=====
```

```
# parameters
```

```
#=====
```

```
infile = 'TransferCurve.csv'
```

```
dg = 100e-9 #m
```

```
erg = 11.9
```

```
W = 300.0e-6 #m
```

```
L = 50.0e-6
```

```
# Ids^(1/2)-VgsプロットをするVds
```

```
Vds0 = 10.0
```

```
# 余計な文字が含まれている文字列から、  
# 浮動小数点に変換できる最初の文字列を切り出し、  
# 浮動小数点に変換して返す
```

```
def pfloat(str, defval = None):
```

```
# 文字列から、浮動小数点に使える文字が連続している部分を切り  
出す
```

```
    m = re.search(r'([+¥-eE¥d¥.]*)', str)
```

```
# 一致した文字列を取得
```

```
    valstr = m.group()
```

```
    try:
```

```
        return float(valstr)
```

```
    except:
```

```
        return defval
```

```
def read_csv(fname):
```

```
    print("")
```

```
    with open(fname) as f:
```

```
        fin = csv.reader(f)
```

```
        labels = next(fin)
```

```
        xlabel = labels[0]
```

```
# label行が 空文字 の場合、データとしては読み込まない
```

```
        ylabels = []
```

```
        for i in range(1, len(labels)):
```

```
            if labels[i] == ":
```

```
                break
```

```
            ylabels.append(labels[i])
```

```
        ny = len(ylabels)
```

```
        x = []
```

```
        ylist = []
```

```
        for i in range(ny):
```

```
            ylist.append([])
```

```
        for row in fin:
```

```
            x.append(pfloat(row[0]))
```

```
            for i in range(1, ny+1):
```

```
                v = pfloat(row[i])
```

```
                if v is not None:
```

```
                    ylist[i-1].append(v)
```

```
                else:
```

```
                    ylist[i-1].append(None)
```

```
    return xlabel, ylabels, x, ylist
```

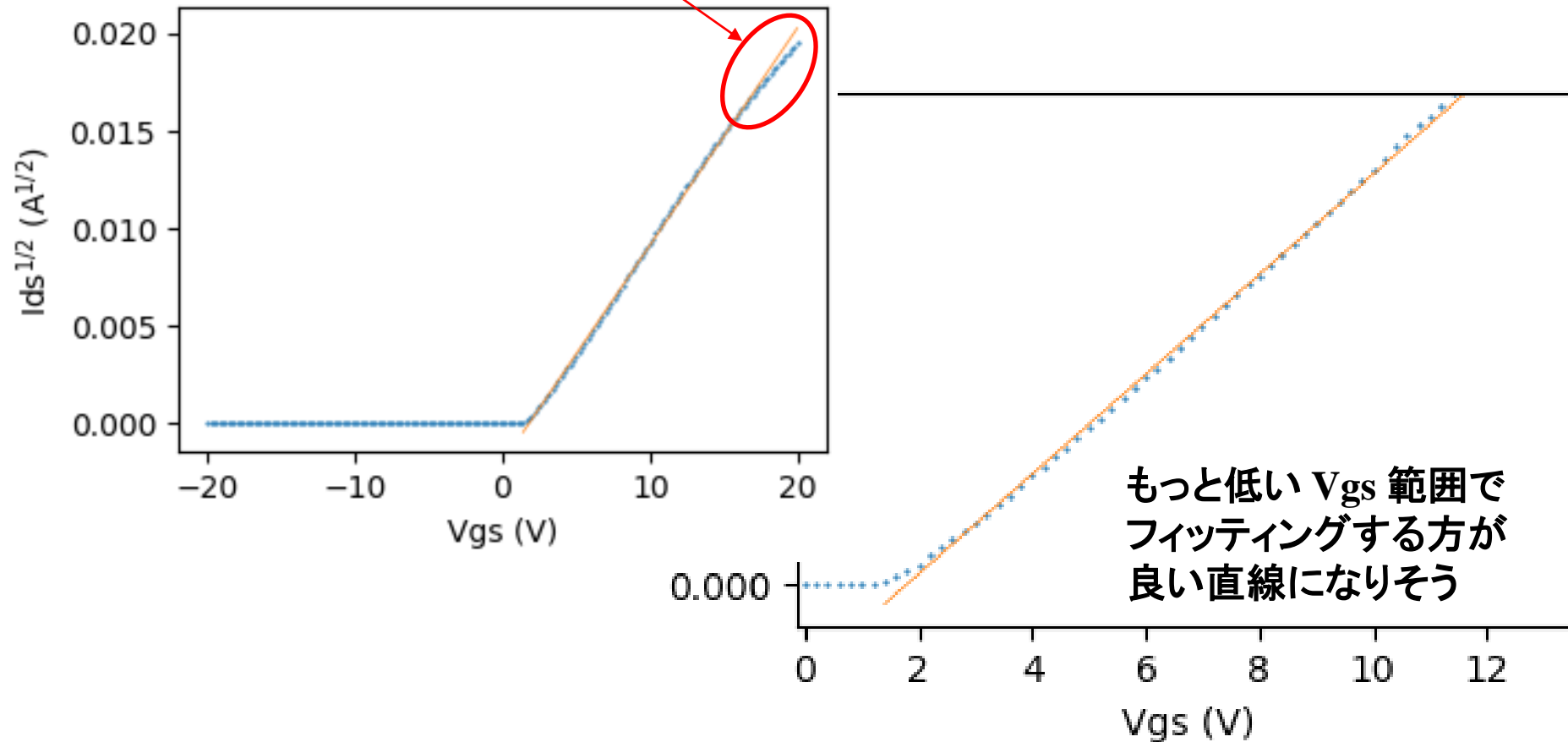
フィッティング範囲の妥当性

TFTiv_errorcheck.py

フィッティング範囲 2.0 ~ 10.0V

高 V_{gs} では $V_{ds} < V_p = V_{gs} - V_{th}$ となり、
直線から外れる

最小二乗に入れてはいけない



線形最小二乗法 $y = a + bx$ の誤差

酒井英行訳、M.C.Barford著、実験精度と誤差、丸善

$$f(x_i) = a + bx_i$$

$$\varepsilon_i = f(x_i) - (a + bx_i) \quad \text{目的関数 } S = \sum \varepsilon_i^2 \text{ を最小化}$$

$$\begin{pmatrix} n & s_x \\ s_x & s_{xx} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} s_y \\ s_{xy} \end{pmatrix} \quad s_x = \sum x_i, s_y = \sum y_i, s_{xx} = \sum x_i^2, s_{xy} = \sum x_i y_i$$

$\langle A \rangle$ は A の平均

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\Delta} \begin{pmatrix} s_{xx} & -s_x \\ -s_x & n \end{pmatrix} \begin{pmatrix} s_y \\ s_{xy} \end{pmatrix} \quad \Delta = ns_{xx} - s_x^2 = n \sum (x_i - \langle x \rangle)^2$$

$$b = \frac{ns_{xy} - s_x s_y}{\Delta} = \frac{\sum (x_i - \langle x \rangle)(y_i - \langle y \rangle)}{\sum (x_i - \langle x \rangle)^2} \quad a = \frac{s_{xx} s_y - s_x s_{xy}}{\Delta} = \langle y \rangle - b \langle x \rangle$$

$$\sigma_y^2 = \langle y^2 \rangle - \langle y \rangle^2 - \frac{(\langle xy \rangle - \langle x \rangle \langle y \rangle)^2}{\langle x^2 \rangle - \langle x \rangle^2} = \frac{1}{n^2} \left[ns_{yy} - s_y^2 - \frac{(ns_{xy} - s_x s_y)^2}{ns_{xx} - s_x^2} \right]$$

$$\text{標準誤差: } S_a = \frac{\sigma_y \sqrt{\langle x^2 \rangle}}{\sqrt{(n-2)(\langle x^2 \rangle - \langle x \rangle^2)}}$$

$$S_b = \frac{\sigma_y}{\sqrt{(n-2)(\langle x^2 \rangle - \langle x \rangle^2)}}$$

$$\text{相関係数: } r = \frac{s_{xy}}{\sqrt{s_{xx} s_{yy}}}$$

誤差の計算

誤差の伝播則

変数 a, b の標準誤差 σ_a, σ_b が既知の場合、 $f(a, b)$ の誤差は

$$\delta f(a, b) = \left(\frac{\partial f}{\partial a} \right)_b \delta a + \left(\frac{\partial f}{\partial b} \right)_a \delta b$$

$\delta a, \delta b$ が正規分布に従う場合、 $\delta f(a, b)$ の標準偏差 σ_f は

$$\sigma_f = \sqrt{\left[\left(\frac{\partial f}{\partial a} \right)_b \sigma_a \right]^2 + \left[\left(\frac{\partial f}{\partial b} \right)_a \sigma_b \right]^2}$$

最小自乗法で $y = a + bx$ の標準誤差 σ_a, σ_b が得られたら・・・

$$\cdot V_{th}(a, b) = -a/b$$

$$\delta V_{th} = -\delta a/b + a\delta b/b^2$$

$$\delta \log V_{th} = \frac{\delta V_{th}}{V_{th}} = \delta a/a - \delta b/b$$

$$\sigma_{V_{th}} = \sqrt{\left[\left(\frac{1}{b} \right) \sigma_a \right]^2 + \left[\left(\frac{a}{b^2} \right) \sigma_b \right]^2}$$

$$\sigma_{V_{th}} = V_{th} \sqrt{\left[\left(\frac{1}{a} \right) \sigma_a \right]^2 + \left[\left(\frac{1}{b} \right) \sigma_b \right]^2}$$

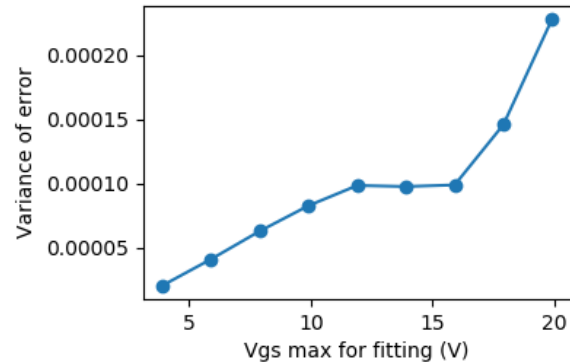
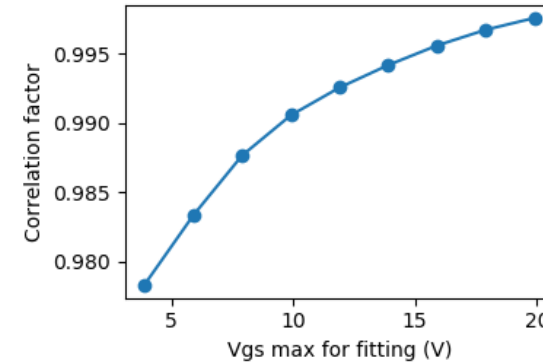
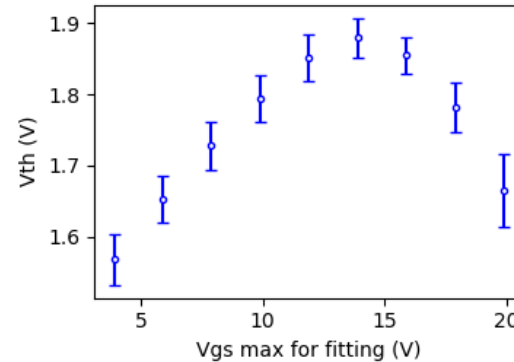
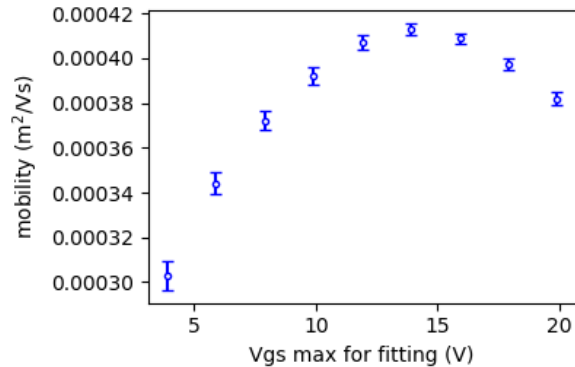
$$\cdot \mu(a, b) = b^2 / \sqrt{\frac{C_{ox}}{2L}}$$

$$\delta \log \mu = \frac{\delta \mu}{\mu} = \delta [\text{const} + 2 \ln b] = 2 \frac{\delta b}{b}$$

$$\sigma_\mu = 2\mu\sigma_b/b$$

最小二乗法の誤差と相関係数

TFTiv_errorcheck.py



- フィッティング範囲を広げると**相関係数**は 1.0 に近づく
x 範囲が広いと相関係数は大きくなりやすい。
必ずしも良い直線範囲の指標にはなっていない
- **誤差 ε_i の標準偏差 σ_{ε_i}** は $V_{gs} > 16V$ で明確に大きくなる:
 $V_{gs} > 16V$ のデータを入れてはいけない
- 移動度、 V_{th} は 14 V までのデータを入れると最大値をとる
意味はない。フィッティング範囲の妥当性の検証が重要
- **標準誤差** は **1σ** で表示していることに注意。
 3σ を見込むのが普通

プログラム (抜粋)

TFTiv_errorcheck.py

誤差、相関係数計算付き 一次多項式線形最小二乗

```
def lsq1(x, y, iPrint = 0):
```

```
    n = len(x)
```

統計量の計算

si: 変数 i の和 avi: i の平均

sij: 変数 i と j の積の和 sij: i*j の平均

```
    sx = sum(x)
```

```
    avx = sx / n
```

```
    sy = sum(y)
```

```
    avy = sy / n
```

```
    sxx = sum([x[i] * x[i] for i in range(n)])
```

```
    avxx = sxx / n
```

```
    sxy = sum([x[i] * y[i] for i in range(n)])
```

```
    avxy = sxy / n
```

```
    syy = sum([y[i] * y[i] for i in range(n)])
```

```
    avyy = syy / n
```

```
    delta = n * sxx - sx * sx
```

$y = a + bx$

```
    b = (n * sxy - sx * sy) / delta
```

```
    a = avy - b * avx
```

残差の二乗和 $\sum \varepsilon_i^2$ 、誤差の標準偏差 $\sigma(\varepsilon_i)$

```
    sum_ei2 = sum([pow(y[i] - a - b * x[i], 2) for i in range(n)])
```

```
    sigma_ei = sqrt(sum_ei2 / (n - 1))
```

```
    sigma_y2 = avyy - avy * avy - pow(avxy - avx * avy, 2) / (avxx - avx * avx)
```

```
    sigma_y = sqrt(sigma_y2)
```

パラメータ a, b の標準誤差と相関係数

```
    Sa = sigma_y * sqrt(avxx) / sqrt((n - 2) * (avxx - avx * avx))
```

```
    Sb = sigma_y / sqrt((n - 2) * (avxx - avx * avx))
```

```
    r = sxy / sqrt(sxx * syy)
```

戻り値が多いので、統計量、標準誤差、相関係数は

辞書変数 (ハッシュ、連想配列) で返す

辞書変数の値は、{key:val}

```
res = {'sx': sx, 'sy': sy, 'sxx': sxx, 'sxy': sxy, 'syy': syy,
```

```
      'Sa': Sa, 'Sb': Sb, 'r': r, 'sigma_ei': sigma_ei, 'residual': sum_ei2}
```

```
return a, b, res
```

```
def main():
```

最小二乗を実行

```
    ai = lsq1(xfit, yfit, 0)
```

辞書変数は ai[2] に入る

```
    res = ai[2]
```

a, b の標準誤差、相関係数

辞書変数の要素は 変数名[key]で受け取る

```
    Sa = res['Sa']
```

```
    Sb = res['Sb']
```

```
    r = res['r']
```

```
    Vth = -ai[0] / ai[1]
```

```
    grad = ai[1]
```

```
    mu = grad * grad / (W * Cox / 2.0 / L)
```

誤差伝播則からVth と μ の標準誤差を計算

```
    SVth = sqrt(pow(Sa / ai[1], 2) + pow(Sb * ai[0] / ai[1] / ai[1], 2))
```

```
    Smu = 2.0 * mu * Sb / ai[1]
```

エラーバー付きグラフのプロット

```
ax.errorbar(xecheck, ymu, yerr = ySmu,
```

```
           capsize = 3.0, fmt = 'o', markersize = 3.0, ecolor = 'b',
```

```
           markeredgecolor = 'b', color = 'w')
```