

Computational Materials Science (計算材料学特論)

Lecture materials updated

http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms

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

Lecture materials for numerical analyses (by Kamiya)
数値解析: 方程式の数値解法, 非線形最適化 (担当分)

We will wait for five minutes.

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

Update News:

- June 27, 00:58 Lecture materials on June 27 have been updated ([20250627EquationOptimize.zip](#))

Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化)

Audio explanation generated by Google NotebookLM: 0:00 / 27:23

Google NotebookLMで生成した音声解説: 0:00 / 9:51

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

Smoothing (平滑化), Interpolation (補間)

Audio explanation generated by Google NotebookLM: 0:00 / 17:33

Google NotebookLMで生成した音声解説: 0:00 / 10:47

- June 17, 12:10 Final version of lecture materials on June 17 have been updated ([20250617Diffeq2.zip](#))

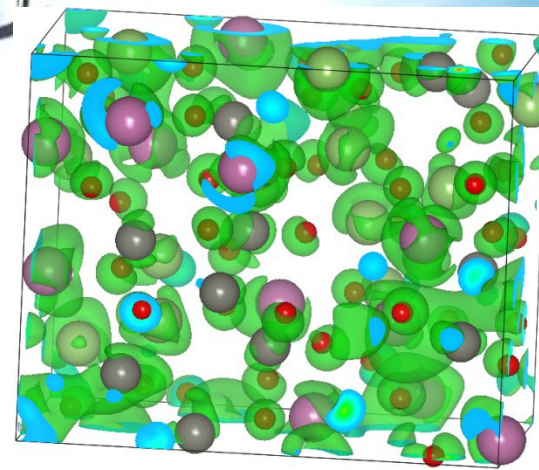
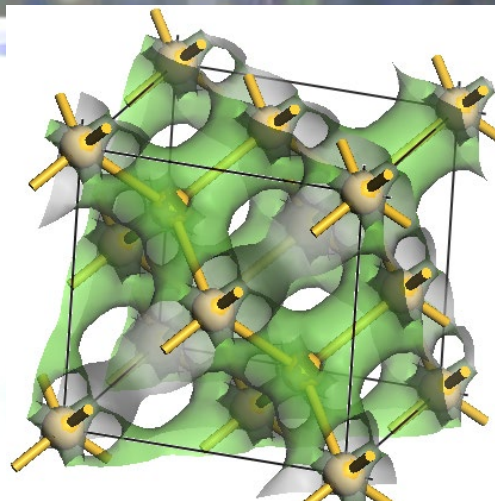
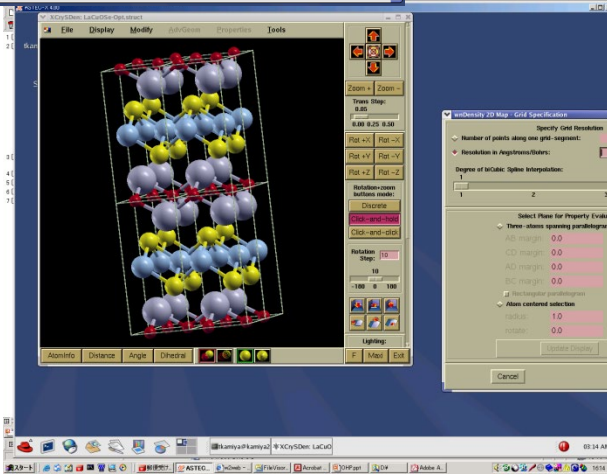
Differential equation (微分方程式), Molecular dynamics (分子動力学法), Interpolation (補間), Smoothing (平滑化)

Audio explanation generated by Google NotebookLM: 0:00 / 15:33

Google NotebookLMで生成した音声解説: 0:00 / 9:10

Also listen the AI-generated brief explanation of the lecture

神谷利夫



New function: Google NotebookLM

http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms

COMPUTATIONAL MATERIALS SCIENCE 2025 Q2

2025年度Q2 計算材料学特論 (資料: 英語 + 日本語版)

Lecture materials for numerical analyses (by Kamiya)




数値解析に関する講義資料・pythonプログラム (神谷担当分)


Google NotebookLM Chatbot

Update News:

- June 26, 06:01 Lecture materials on June 27 have been uploaded ([20250626EquationOptimize.zip](#))

Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化)

Audio explanation generated by Google NotebookLM:  0:00 / 23:34  


Google NotebookLMで生成した音声解説:  0:00 / 9:29  


- June 20, 11:06 Final version of lecture materials on June 20 have been updated ([20250620InterporlateSmoothing2.zip](#))


Smoothing (平滑化), Linear least-squares method (線形最小二乗法), Numerical solutions of equations (方程式の数値解法)


New function: Google NotebookLM


URL may change: Please access from the CMS web page

 Computational Materials Science 2025


 アナリティクス


 新機能！一般公開で共有する

 共有

 設定

PRO





ソース

チャット

Studio

Computational Materials Science 2025

5 ソース

These academic materials, likely from a course on **Computational Materials Science**, provide a comprehensive overview of fundamental numerical methods. The first document introduces **computer basics** and various **sources of errors** in computation, such as round-off and truncation errors, emphasizing the importance of careful handling of floating-point numbers. It also covers **numeric representation** in different bases. The foundation, explaining **numerical differentiation** and **integration** techniques well as more advanced methods like Gauss-Legendre and Romberg integration to functions with anomalies. Finally, the materials briefly touch upon machine learning techniques for data analysis.

NotebookLM only allows a single output language to be set for all notebooks, so the output language may vary between English and Japanese depending on the situation.

入力を開始します...

5 個のソース



How do different numerical methods for differentiation and integration affect accuracy and error propagation in computational materials science?

What fundamental





ソース

チャット

Studio



How I can solve a differential equation with high precision?

To achieve high precision when solving a differential equation, you should consider the choice of numerical method, the step size used in the calculation, and careful handling of floating-point errors [1](#) [...](#).

Here are key approaches based on the sources:

- **Choose Higher-Order Numerical Methods** Higher-order methods generally offer better accuracy for a given step size compared to simpler methods [3](#) [...](#).
 - **Euler Formula (オイラー法)**: This is the simplest method, but its accuracy is "not good," and it is "asymmetric" with respect to time, which can lead to larger errors [2](#).
 - **Heun Formula (ホイン法)**: This method improves upon the Euler formula by averaging approximations, leading to better accuracy. It uses an estimate of the next step from the Euler formula to refine the calculation [2](#) [6](#).
 - **Runge-Kutta Formulas (ルンゲークッタ公式)**: These are highly recommended for achieving higher precision. The sources specifically mention 3-stage 3-order and 4-stage 4-order Runge-Kutta formulas [3](#) [...](#). Plots comparing various methods demonstrate that **Runge-Kutta methods, particularly the 3rd and 4th order, exhibit significantly higher accuracy** and better convergence over time than Euler and Heun methods [4](#) [5](#).
 - **Methods for Second-Order Differential Equations**: For second-order differential equations, which can be broken down into two simultaneous first-order equations, methods like Verlet, velocity Verlet, and Leap Frog are discussed [9](#) [...](#).

Class Schedule

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

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

Explanation of the answers

課題解答の解説

PROBLEM, June 20

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.

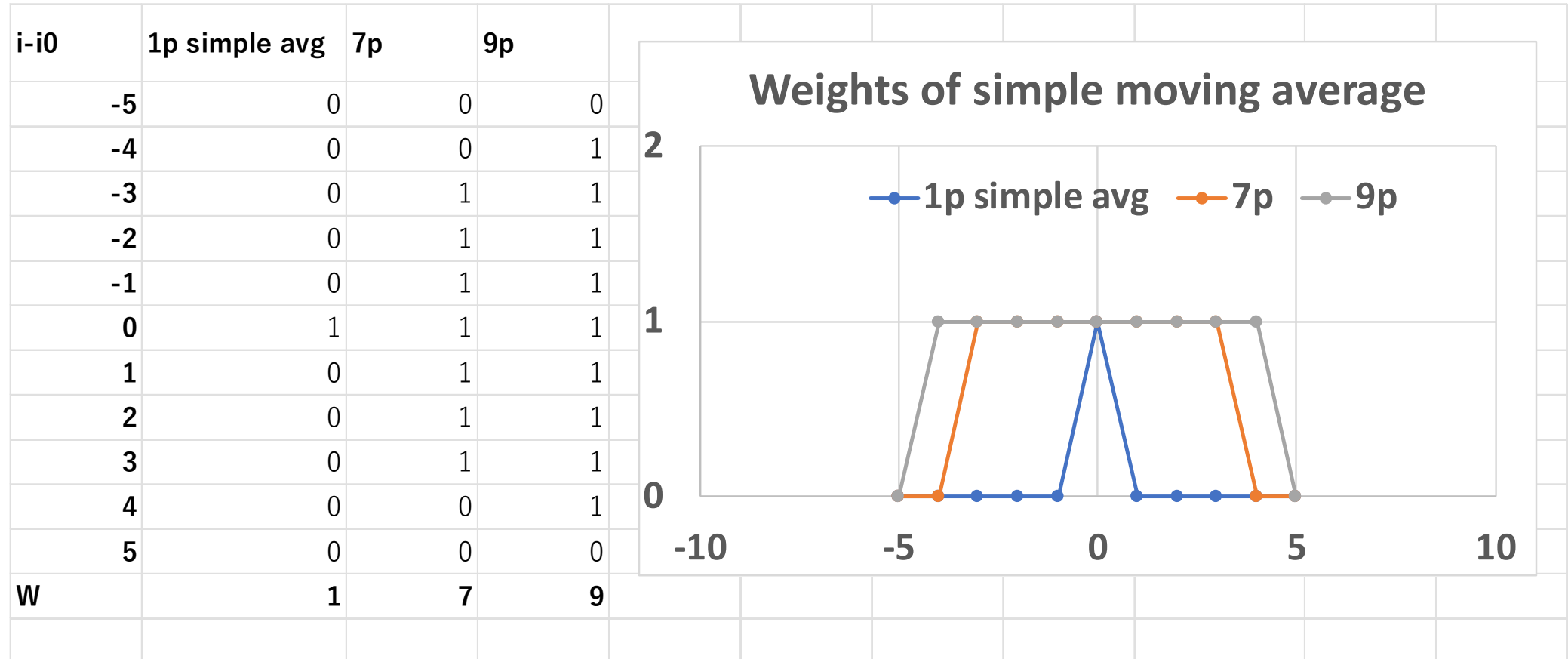
See [dos_smoothing_answer.xlsx](#)

Smoothing

Simple moving average (2m+1 points)

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

Weight w_j/W



Smoothing

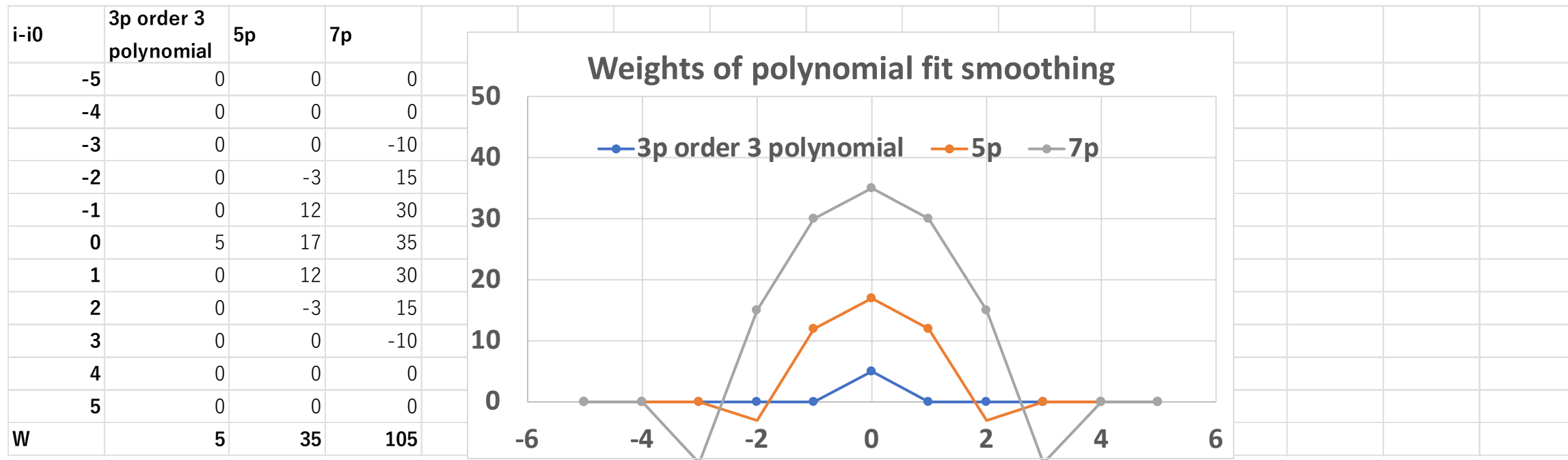
Order 2 and 3 polynomial fit using $(2m+1)$ points

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

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

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

Weight $w_{23}(j)/W$



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 27

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

PROBLEM:

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

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

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

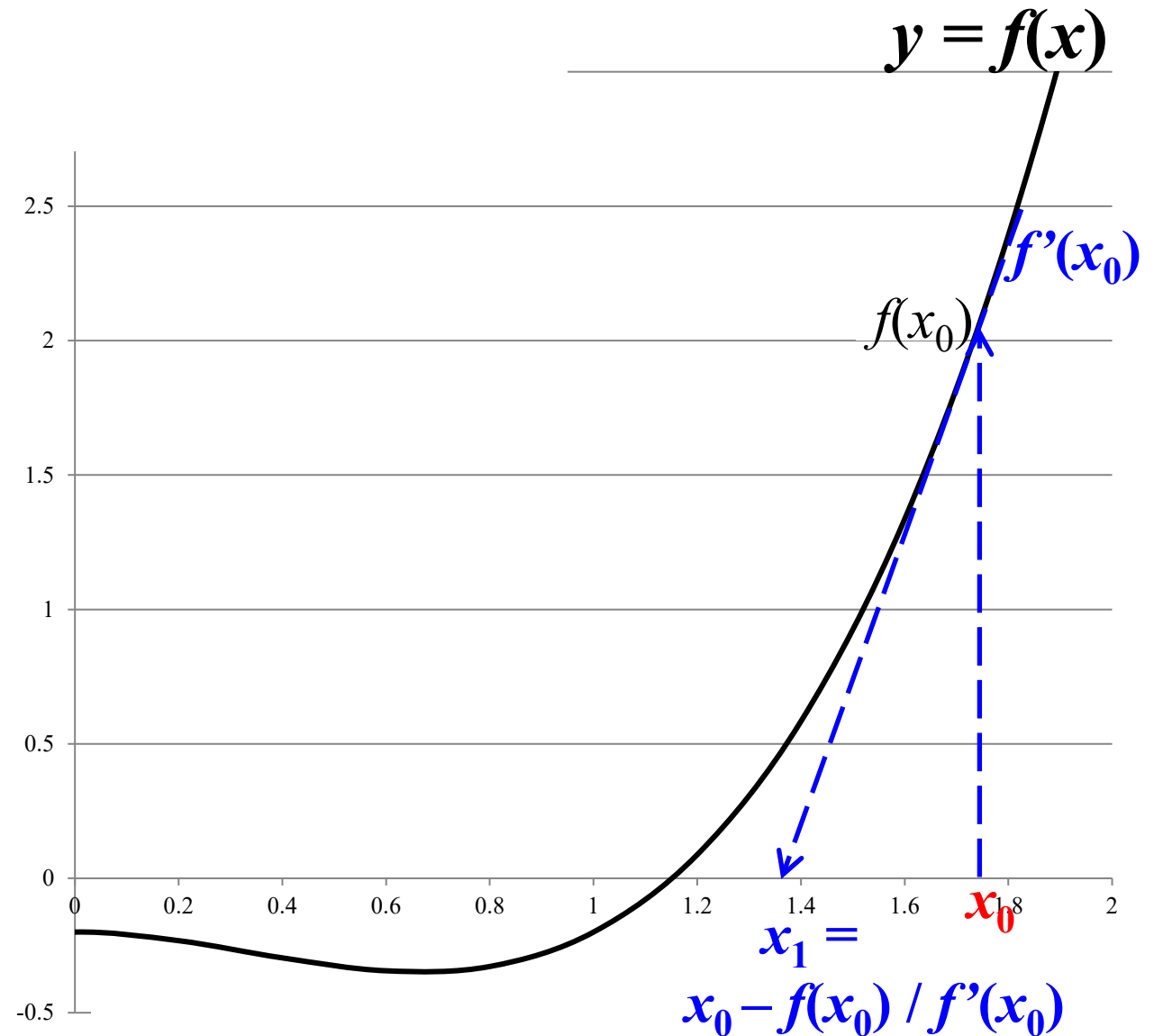
$$f(x_0 + dx) = f(x_0) + dx f'(x_0) \sim 0$$

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

Stabilize convergence:

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

λ : Dumping Factor



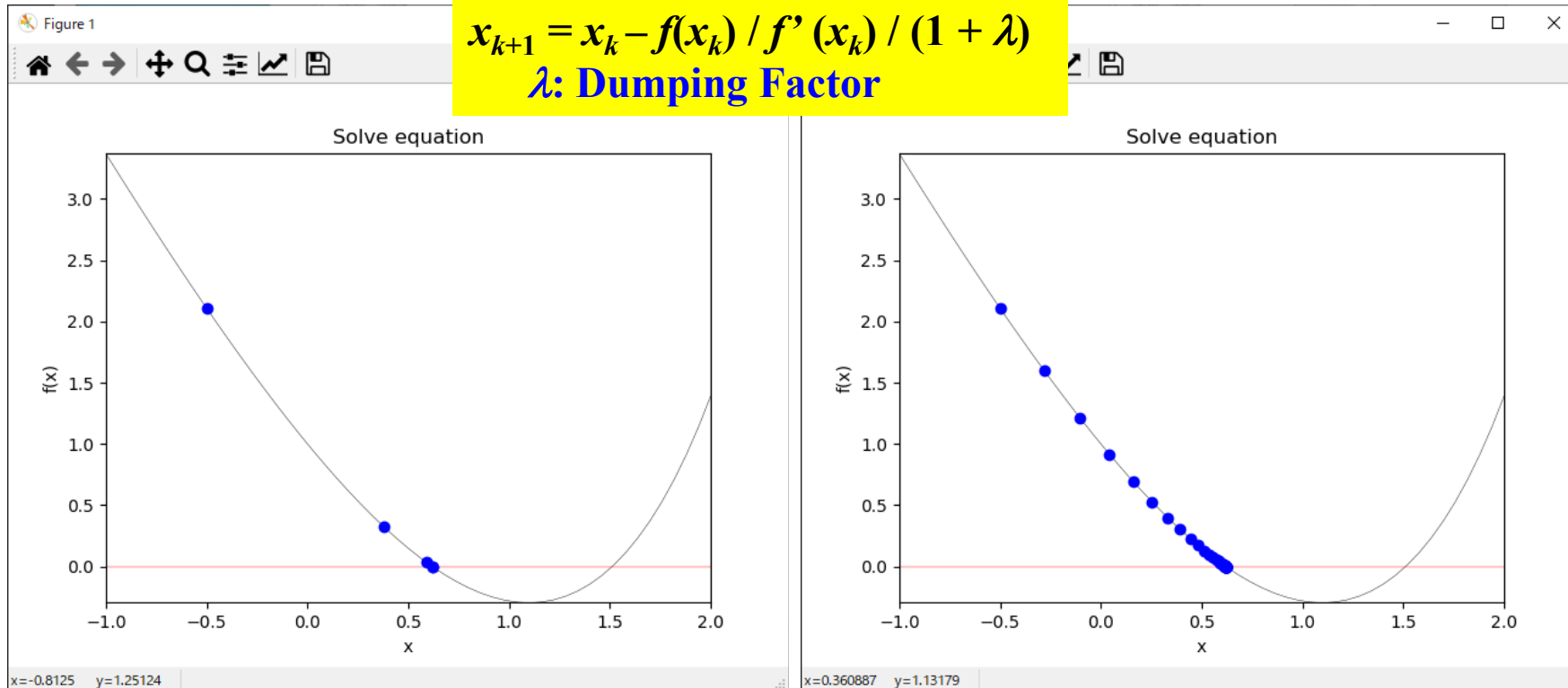
Program: equation-newton-Raphson.py

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



Newton-Raphson method / Secant method

Solve $f(x) = 0$

Start from initial guess: x_0

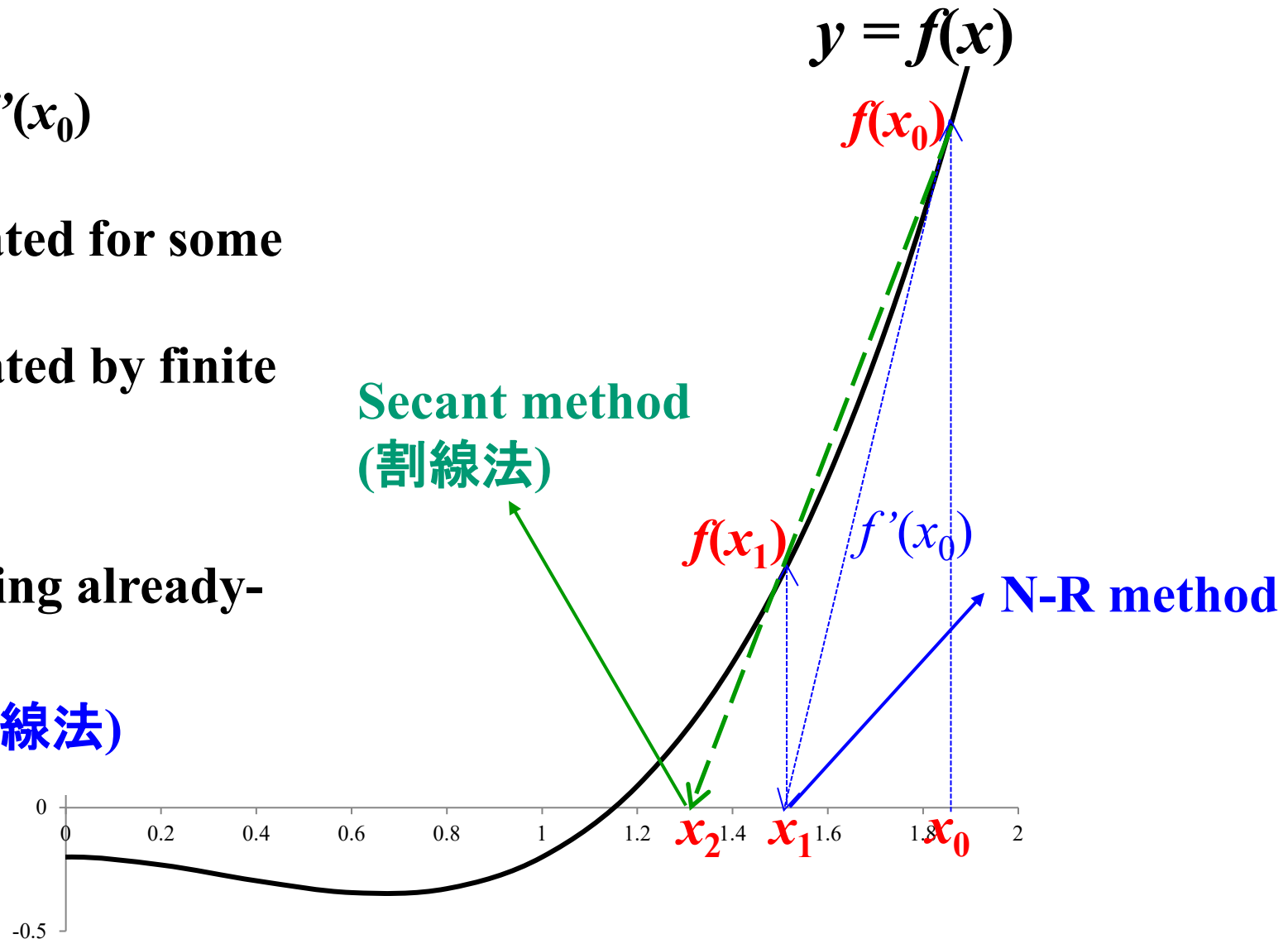
repeat: $x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$

- $f'(x_0)$ can be analytical calculated for some cases
- $f'(x_0)$ is more often approximated by finite difference

$$\frac{f(x_0+h) - f(x_0-h)}{2h}$$

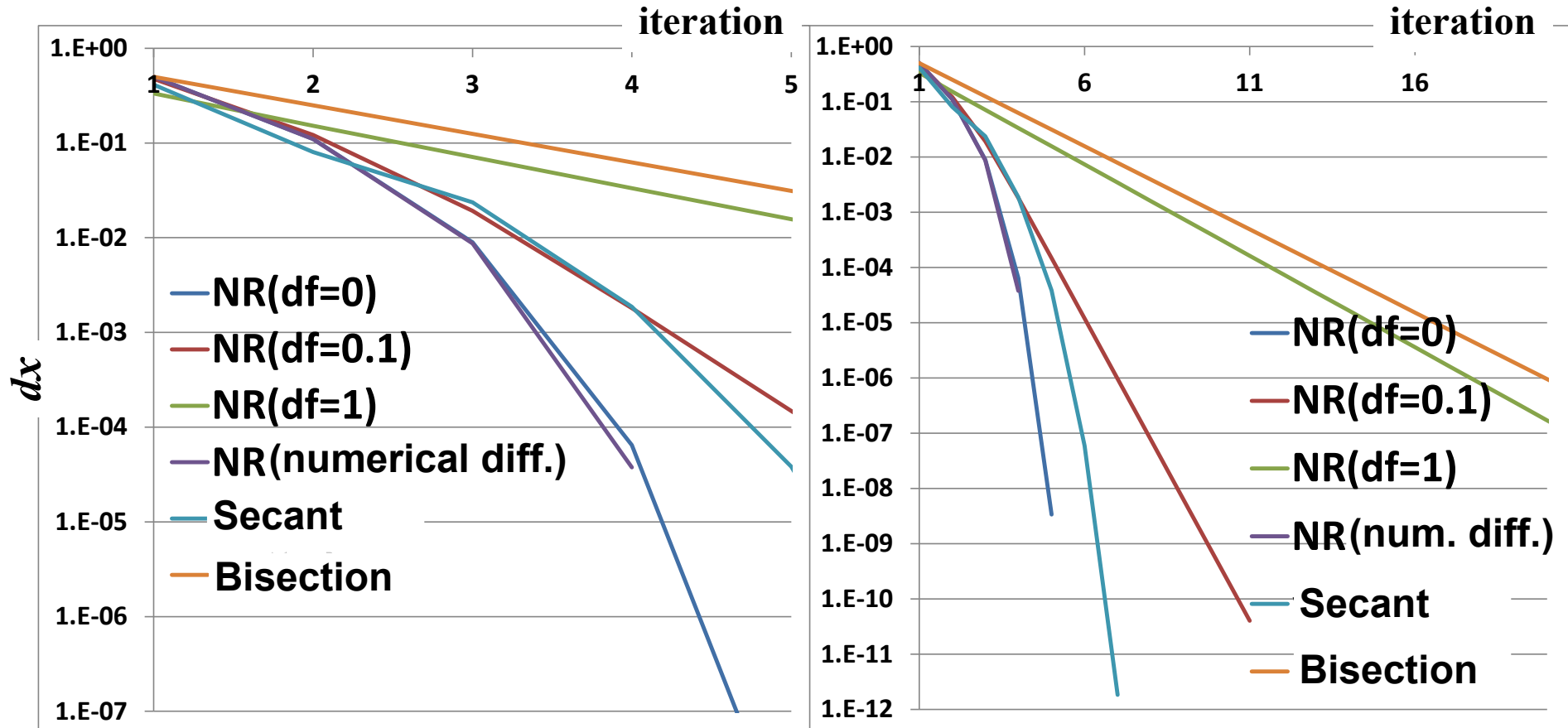
- $f'(x_0)$ can be approximated using already-calculated values

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad \text{Secant method (割線法)}$$



Effect of dumping factor: Convergence process

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



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

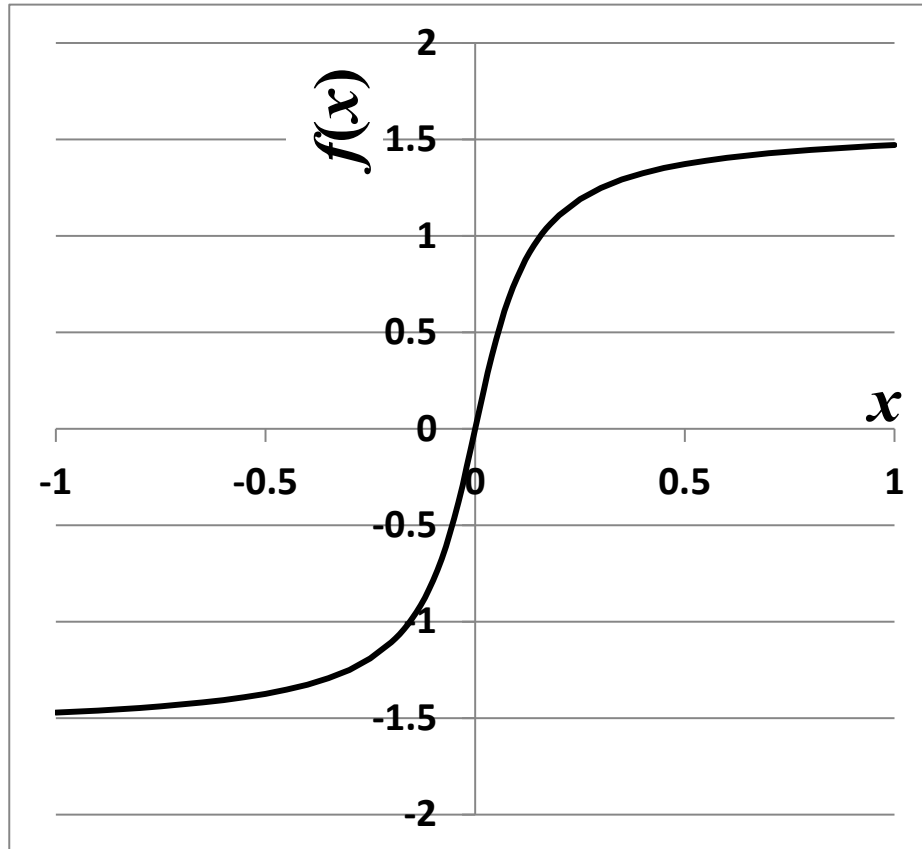
 λ : Dumping Factor

NR: Newton-Raphson method
df: Dumping Factor

Case Newton method succeeds

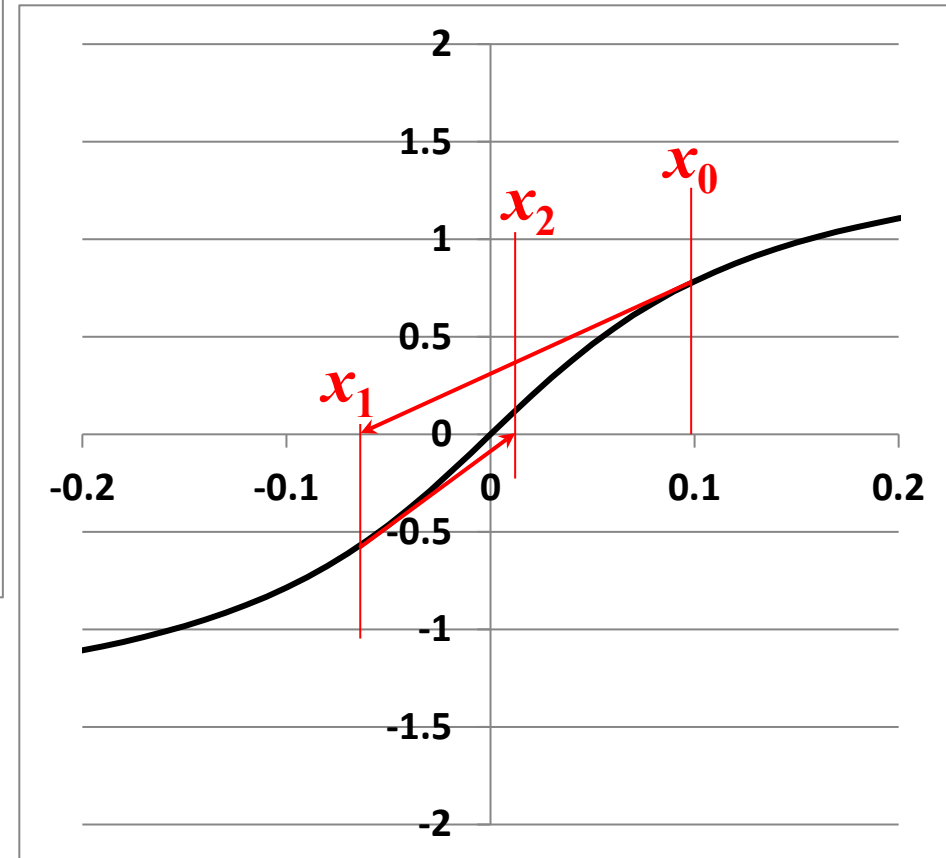
$$f(x) = \tan^{-1}(10x)$$

initial $x = 0.1$



A case to reach convergence

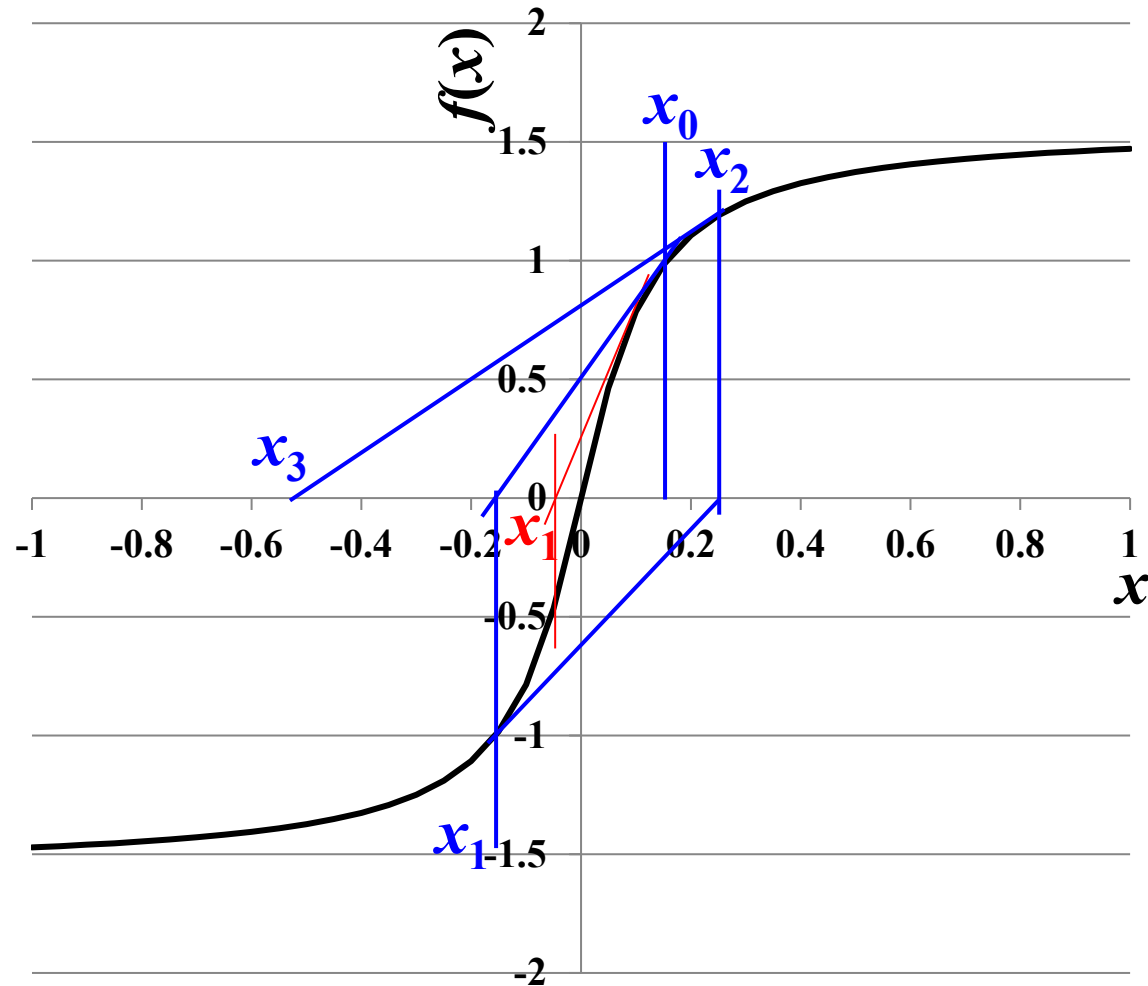
i	x	f(x)	df/dx	dx
0	0.1	0.7854	5	-0.1571
1	-0.05708	-0.5187	7.54257	0.06877
2	0.011686	0.11633	9.86527	-0.0118
3	-0.00011	-0.0011	9.99999	0.00011
4	1.15E-10	1.2E-09	10	-1E-10



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 : Damping 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: Calculate electron density n_e from E_F 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

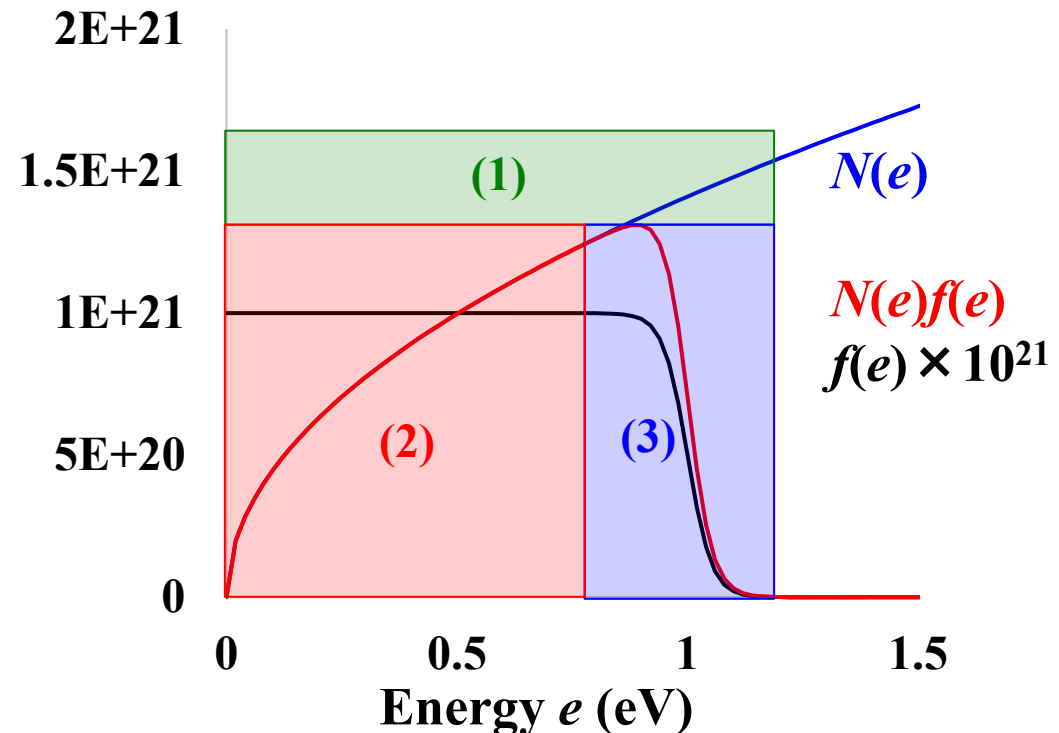
Time is measured for 300 cycles calculation

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 from n_e 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.

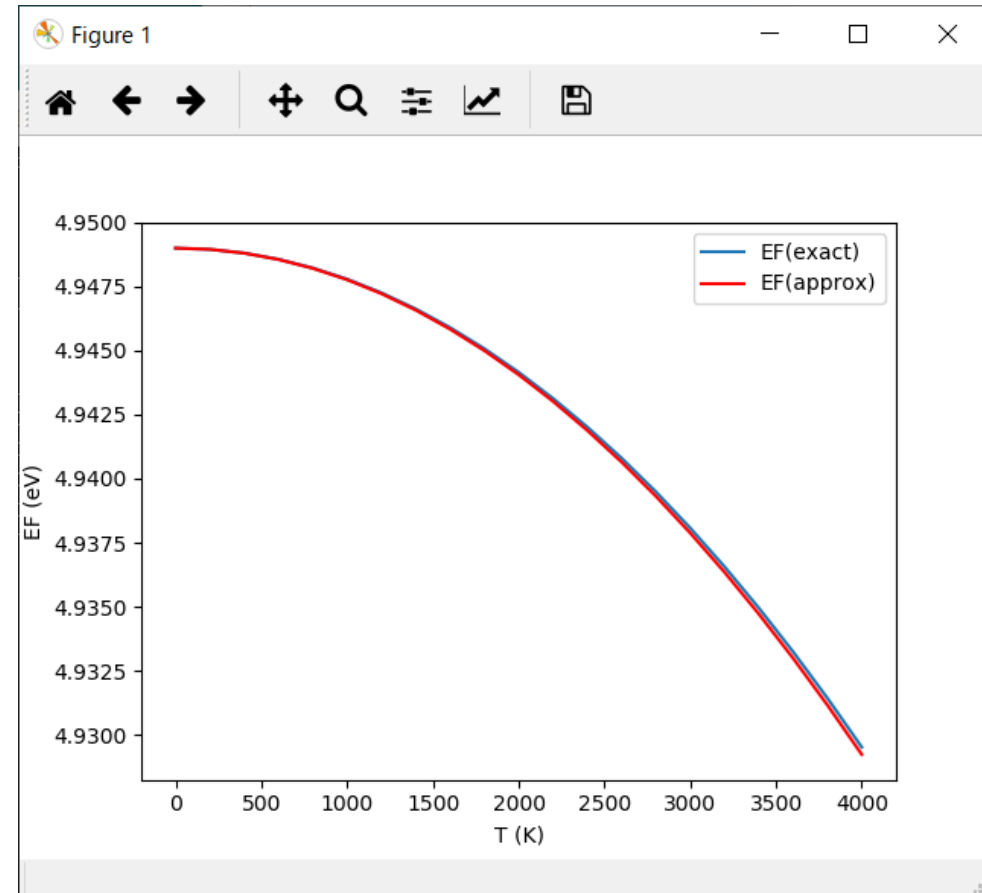
Compare with the approx. form $E_F(T) = E_F(0) - \frac{\pi^2}{6} (k_B T)^2 N'(E_F(0))/N(E_F(0))$

Program: EF-T-metal.py

Ex.: python EF-T-metal.py

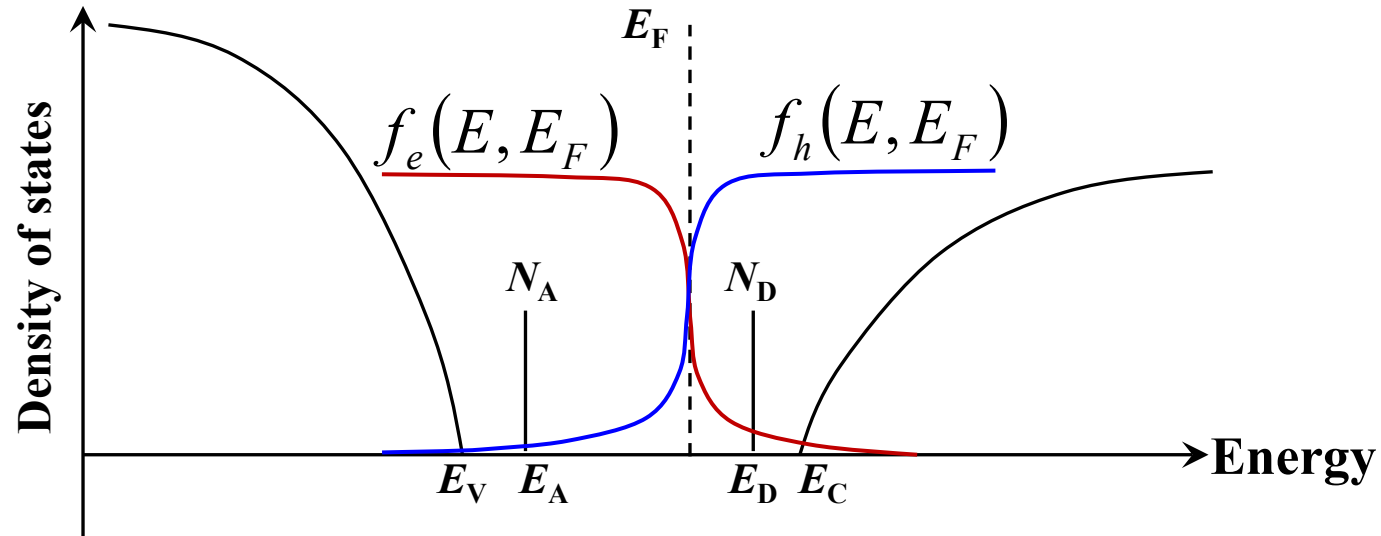
For $n_e = 5 \times 10^{22} \text{ cm}^{-3}$

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

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

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

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

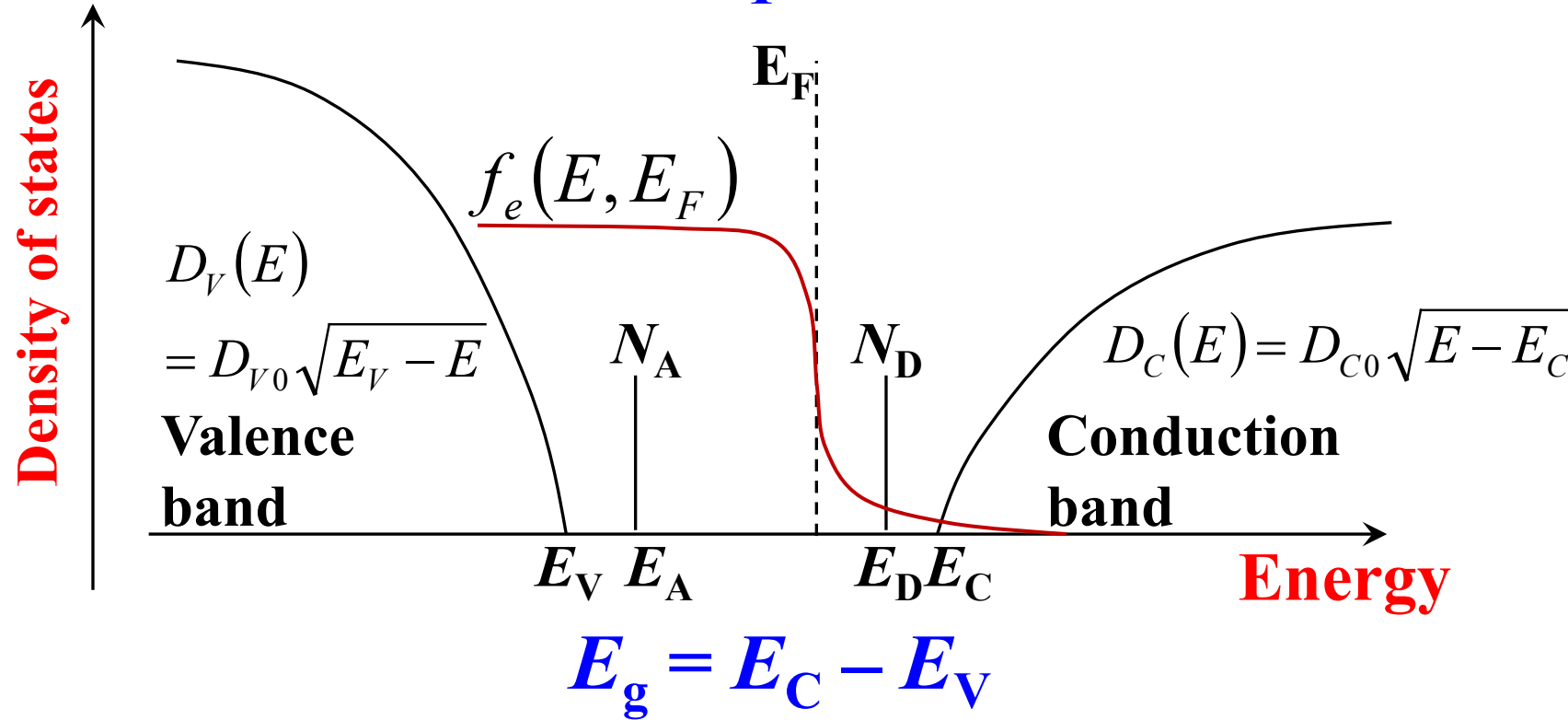
Free electron density

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

Ionized donor density

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

How to determine E_F for semiconductors



Charge neutrality condition

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

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

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

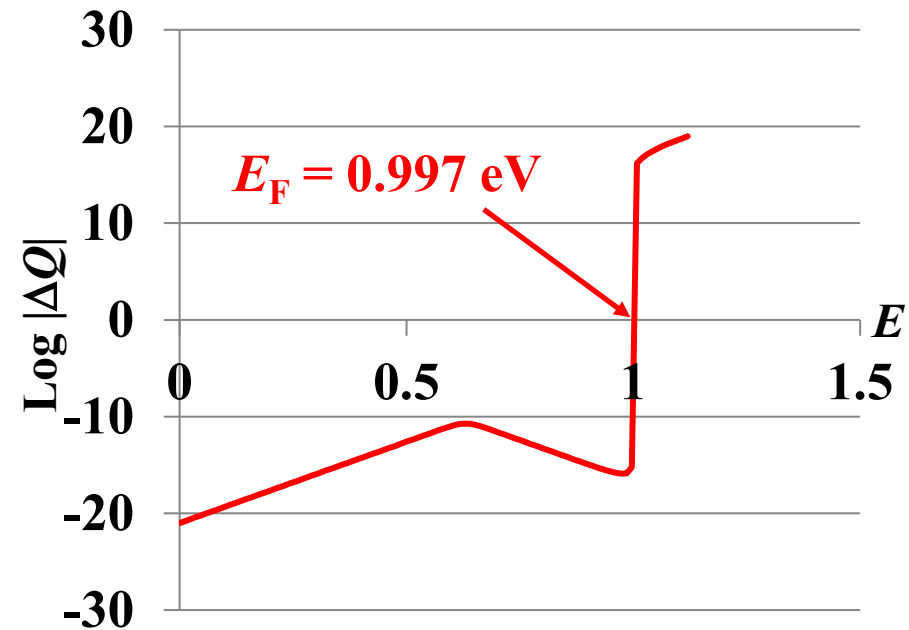
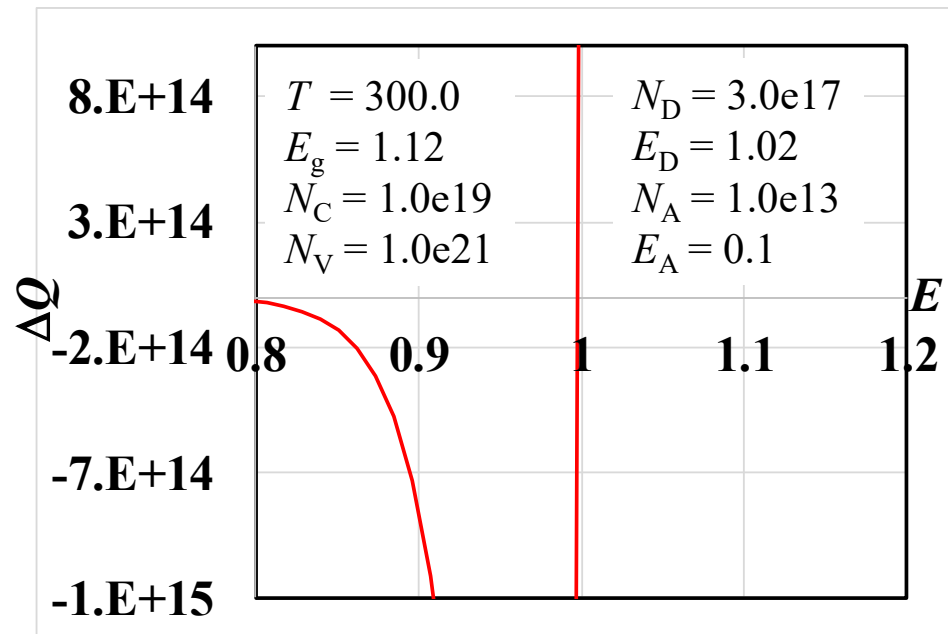
How to calculate E_F : Illustrative solution

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

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

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

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 $f(x)$

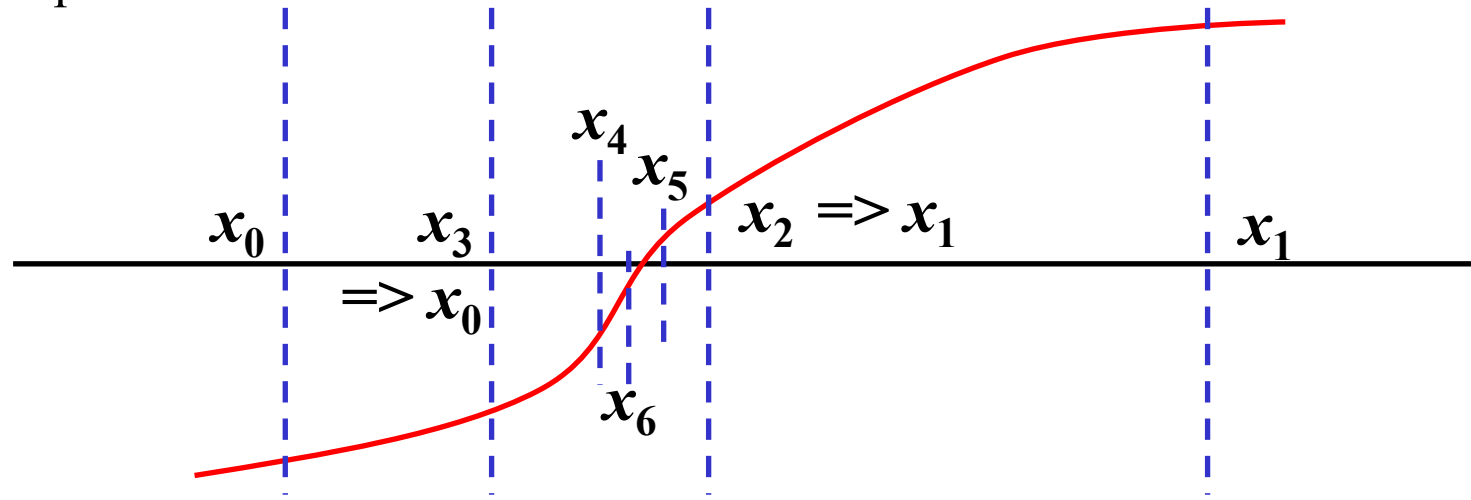
- 1. Start from a range $[x_0, x_1]$ where $f(x_0) < 0$ & $f(x_1) > 0$
(or $f(x_0) > 0$ & $f(x_1) < 0$)**

*** 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 $f(x_0) \cdot f(x_1) < 0$

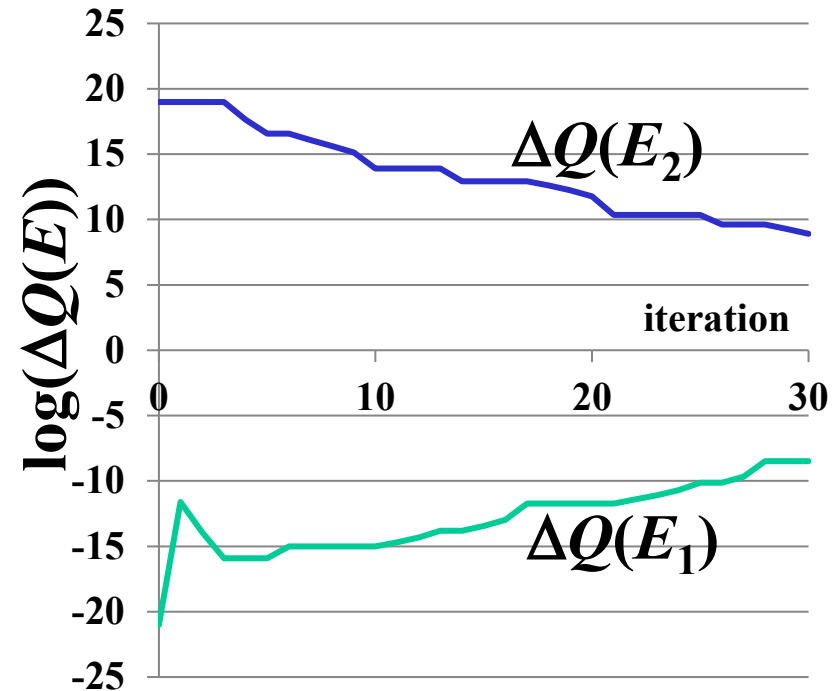
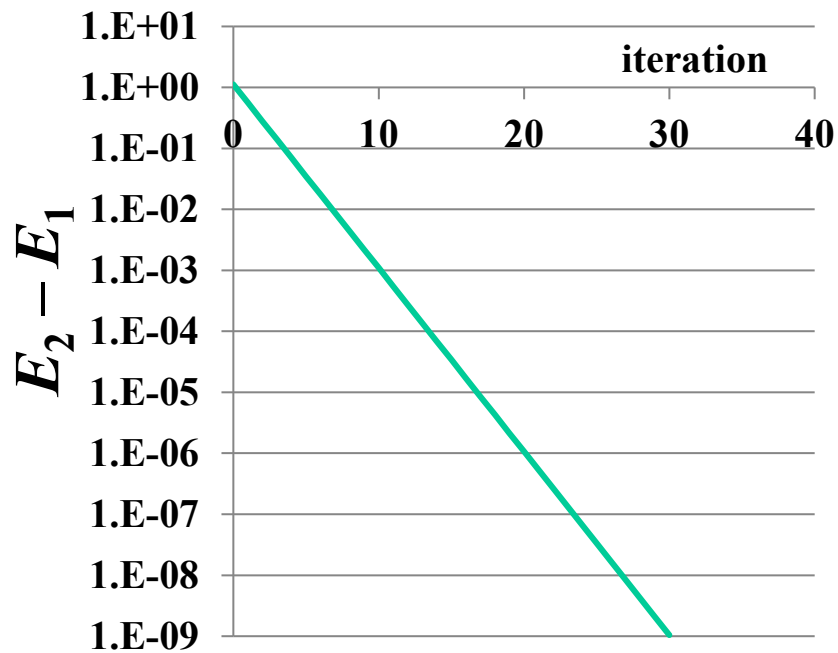
1. $x_2 = (x_0 + x_1) / 2.0$
2. If $f(x_2) > 0$ ($f(x_0) \cdot f(x_2) < 0$), x_1 is replaced with x_2
 If $f(x_2) < 0$ ($f(x_1) \cdot f(x_2) < 0$), x_0 is replaced with x_2
3. Solution x_2 is obtained when $|x_1 - x_0|, |f(x_1) - f(x_0)|$ becomes less than EPS.
4. Repeat 1 – 3



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`

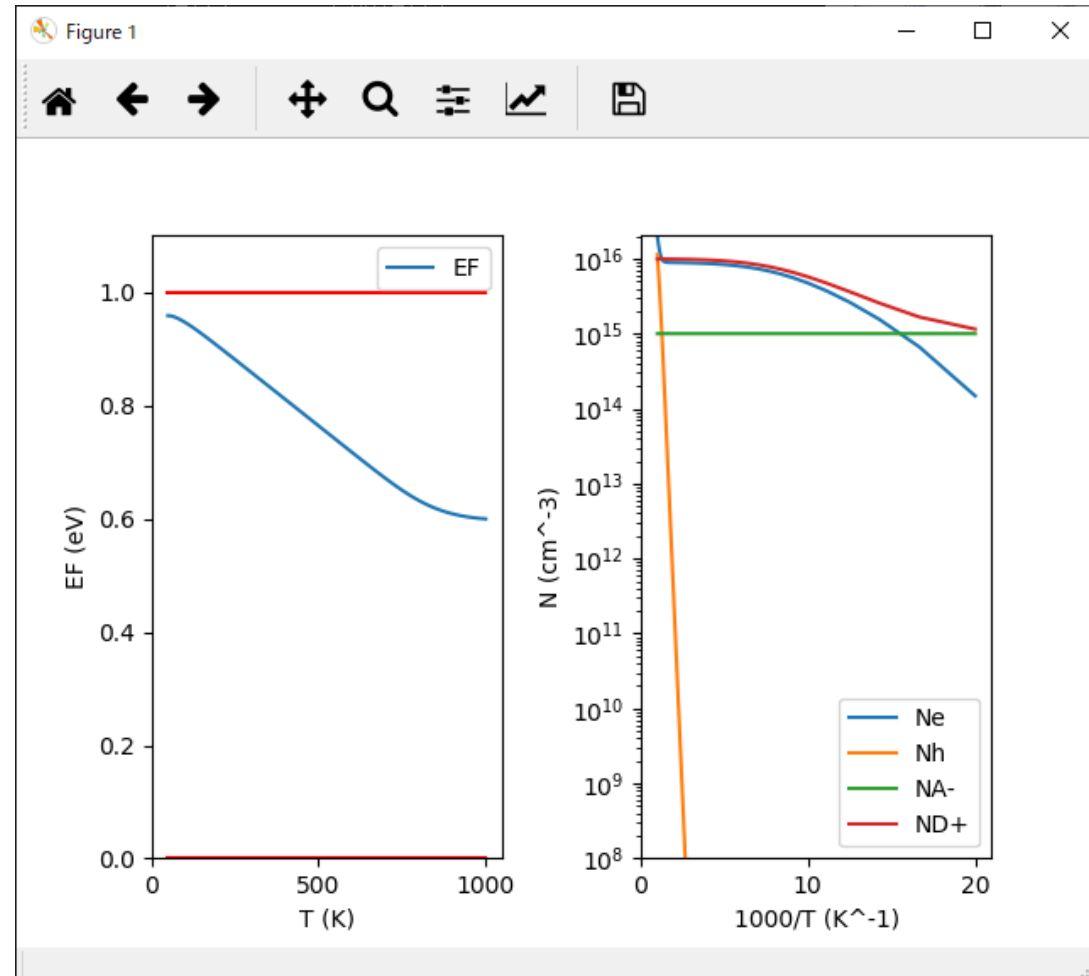
$E_c = 0$, $E_v = 1.0$ eV (= band gap)

$E_A = 0.05$ 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



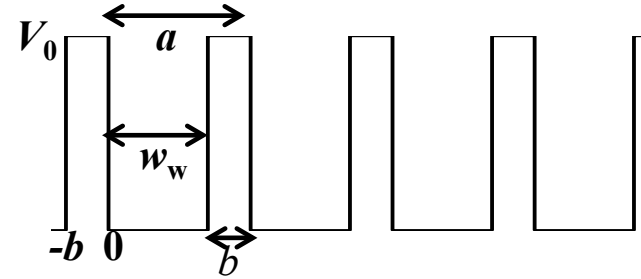
Multi-values equation: Kronig-Penney model

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

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



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:

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

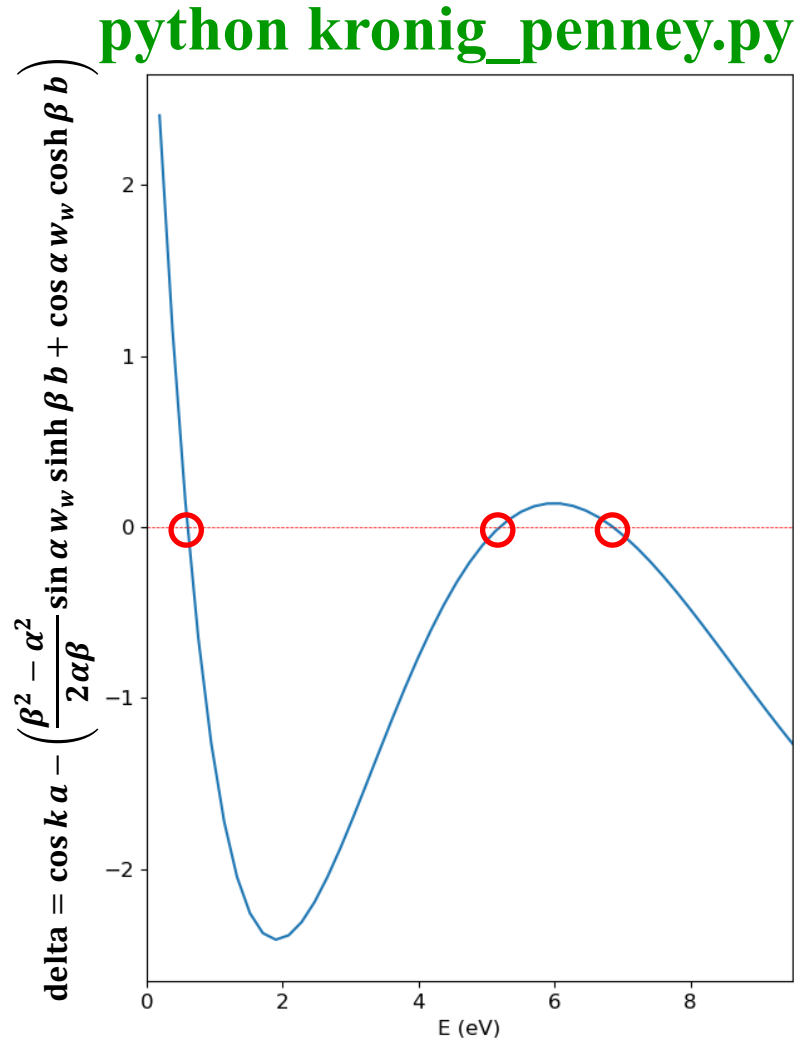
**Scan E in possible range to find all the solutions,
then use them for initial values
to obtain accurate values by Newton-Raphson method**

Program: Kronig-Penney model

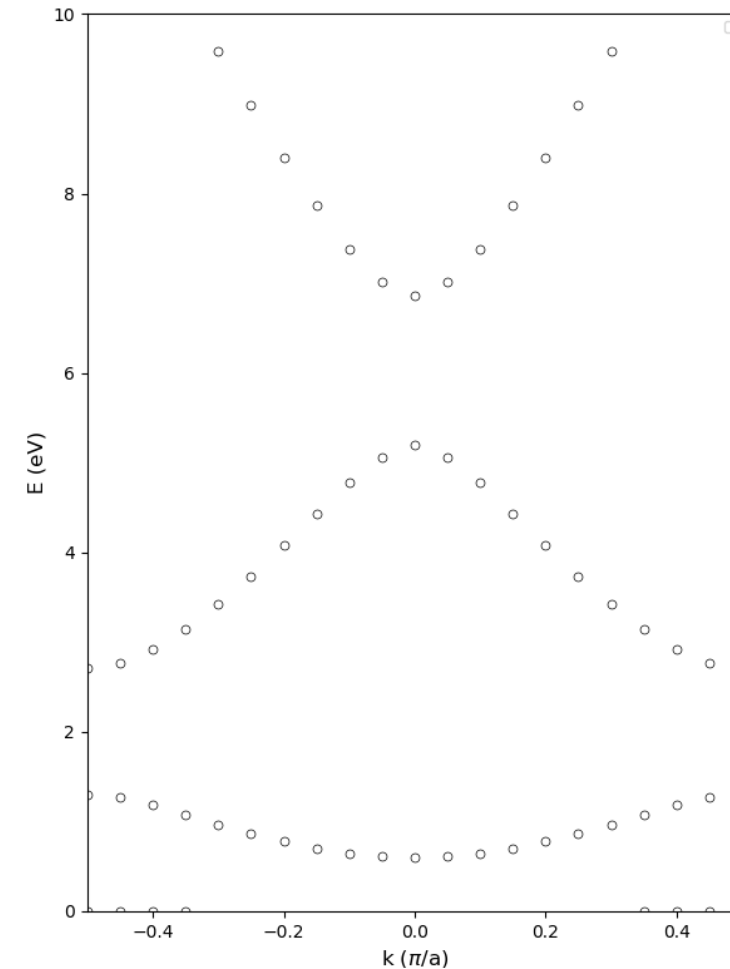
Program: `kronig_penney.py`

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 an objective function $F(\mathbf{x}_i)$

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

Focus on minimization problems

Examples:

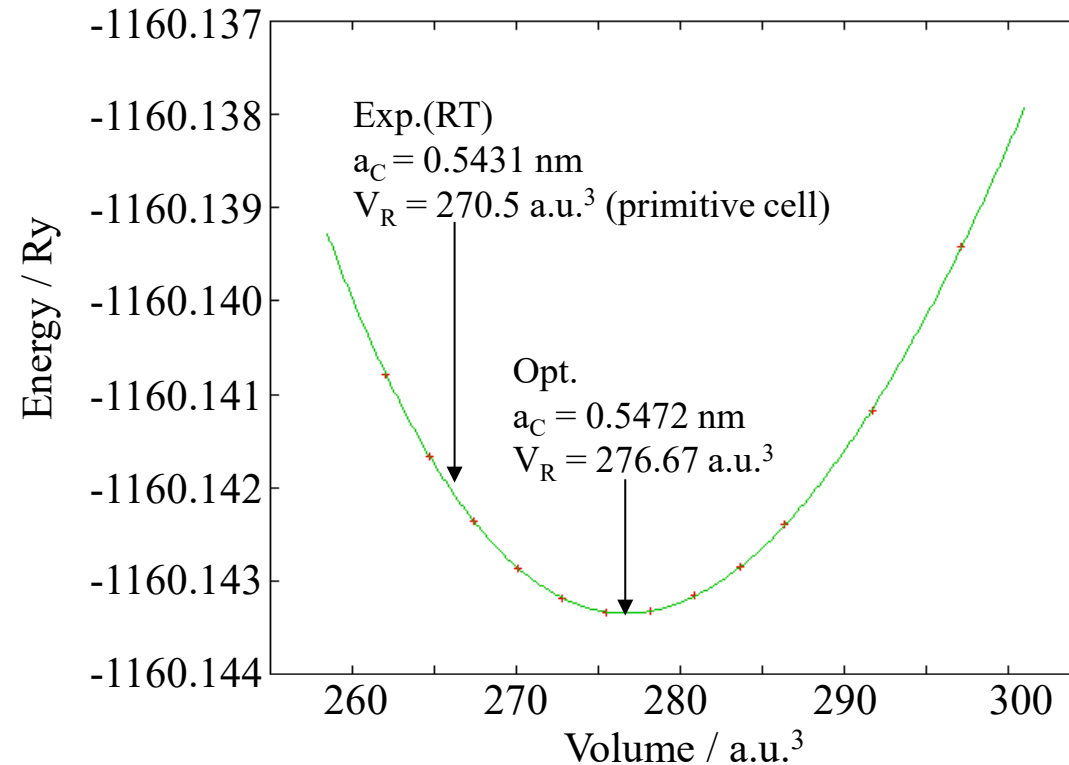
- **Linear least-squares method:** A linear minimization problem for L2 norm of errors
- **Curve fitting:** A non-linear minimization problem for L2 norm of errors
- **Structure relaxation:** A non-linear minimization for total energy

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



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

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

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 $I_G(x)$

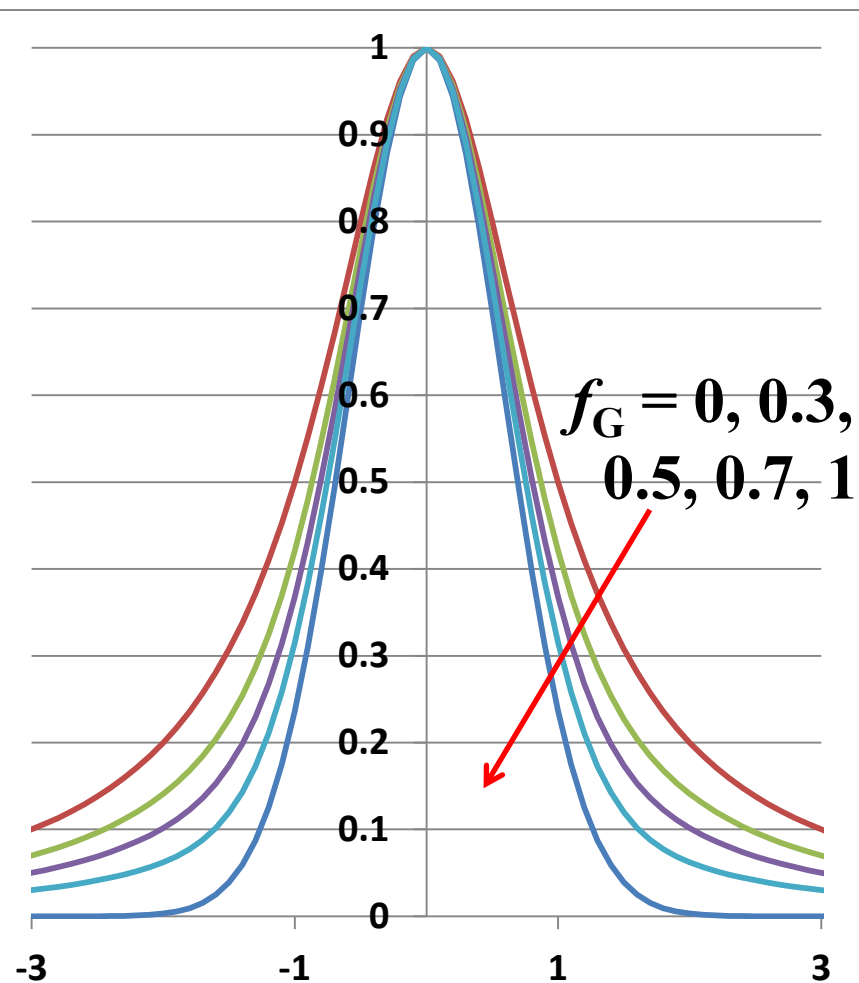
$$I_V(x) = \int_{-\infty}^{\infty} I_G(x') I_L(x - x') dx'$$
$$= \frac{a_V}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x'^2)}{a_V^2 + (x - x')^2} dx'$$

Pseudo-Voigt function:

Simplified Voigt function

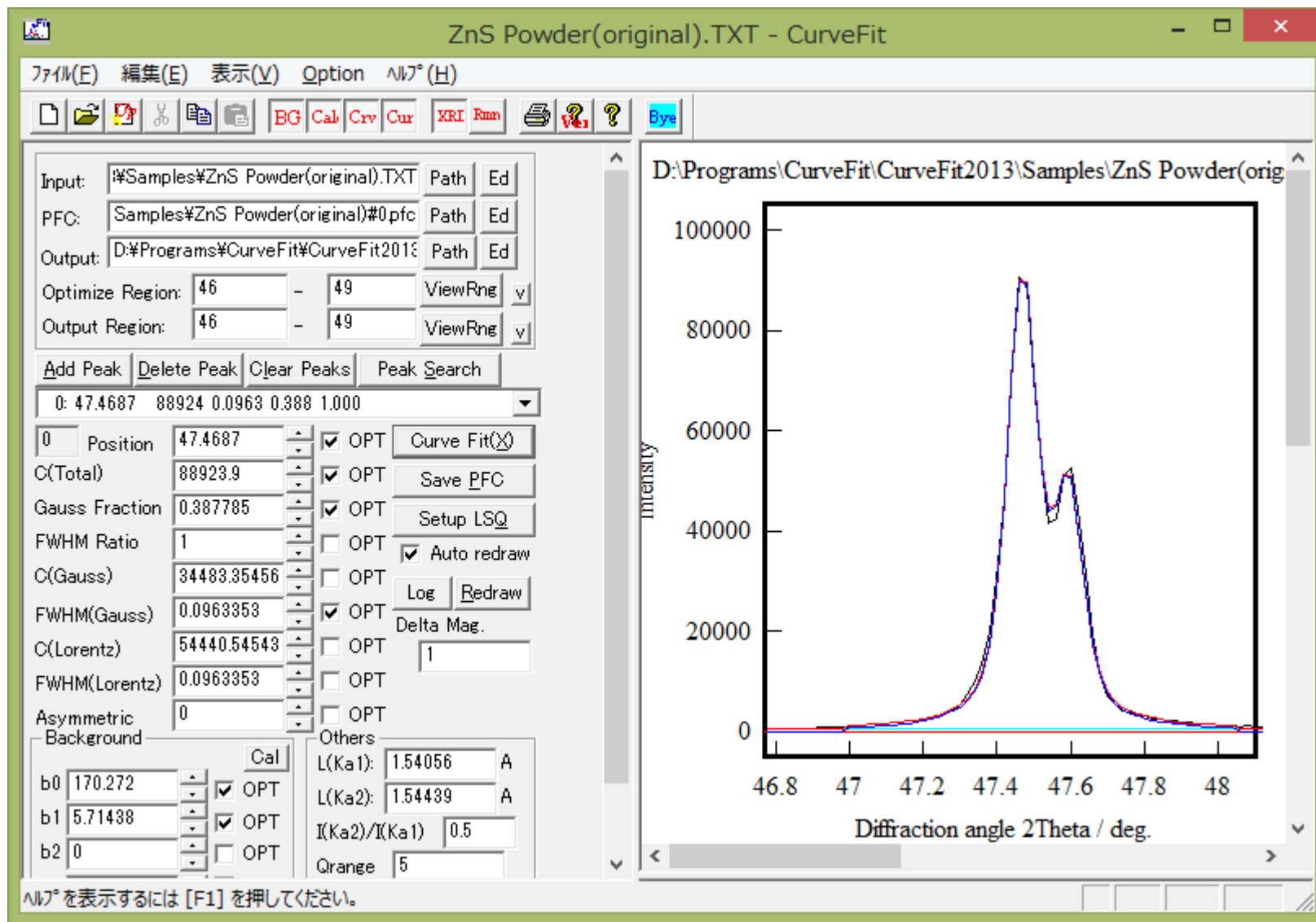
$$I_{PV}(x) = f_G I_G(x) + (1 - f_G) I_L(x)$$

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** $F(x)$:

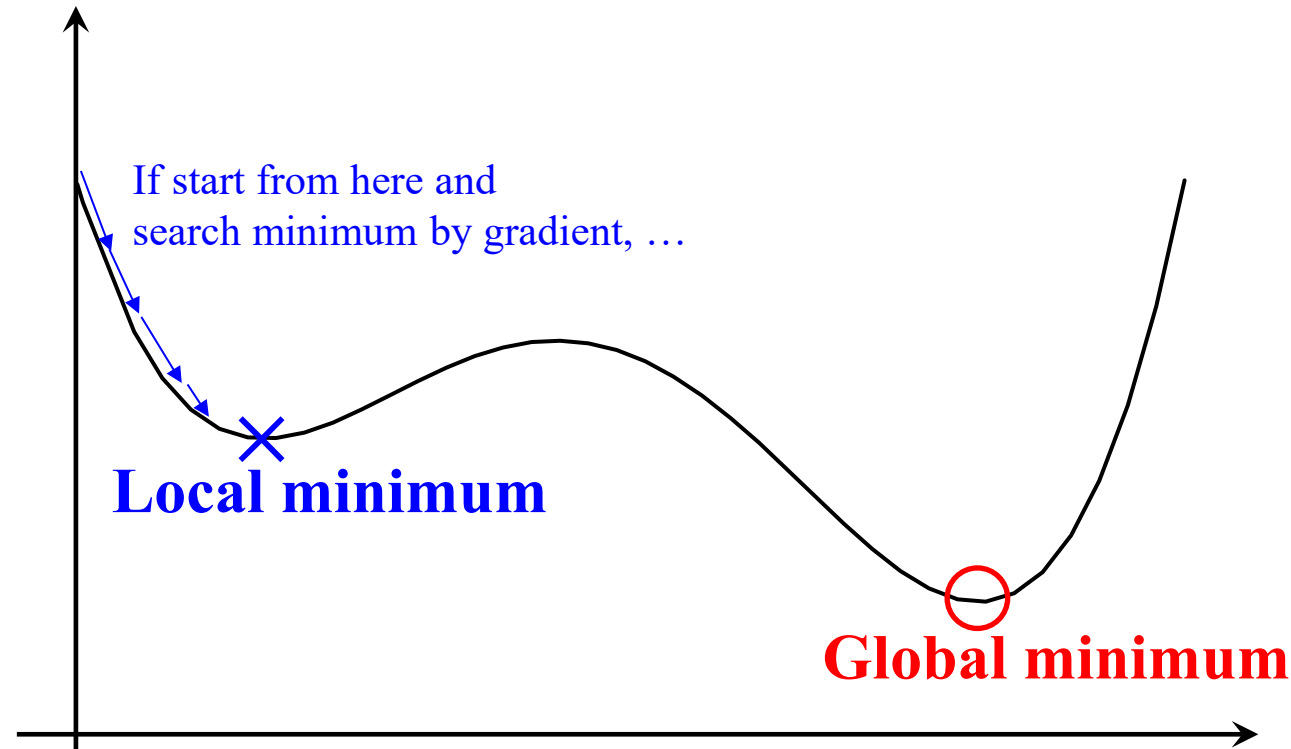
Direct search method (直接探索法)

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:

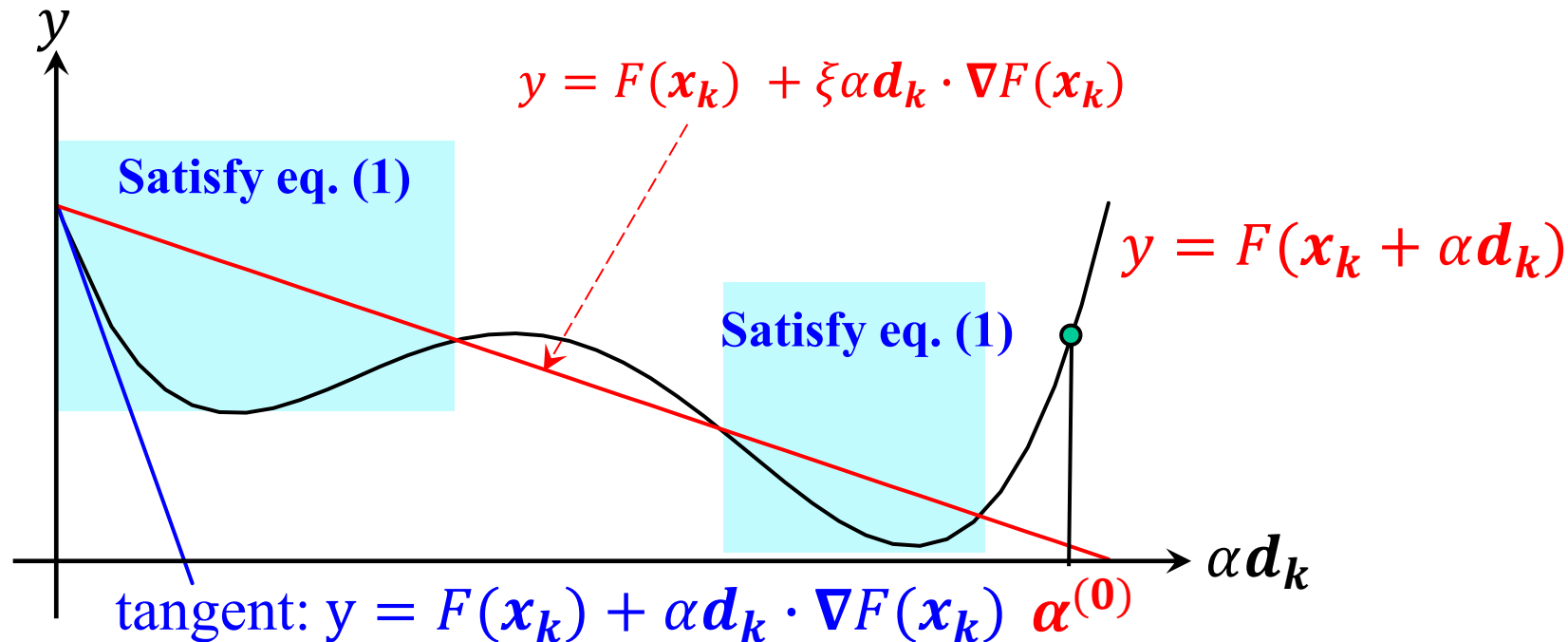
1. **Employ a large initial search range**
2. **Not use a direct value of gradient**

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. $\alpha = \beta_{k,i}$

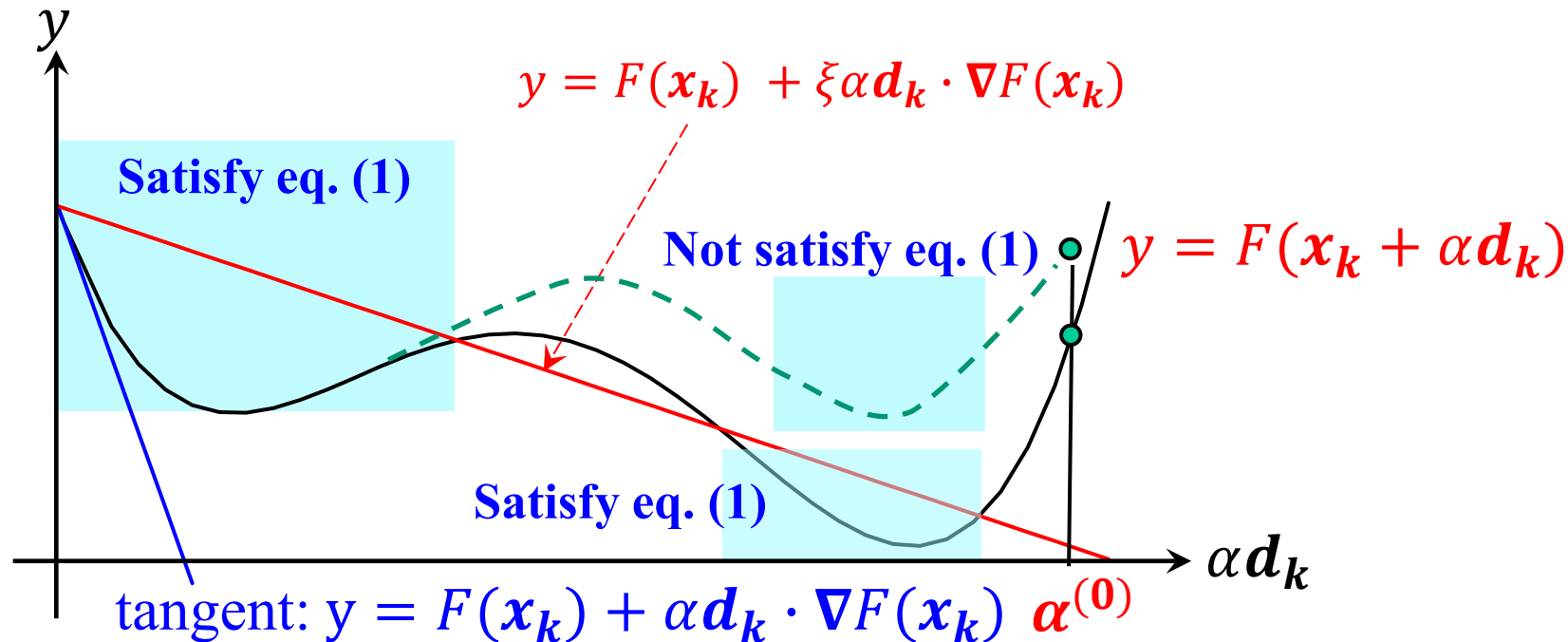


Line search (直線探索法): Armijo condition

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

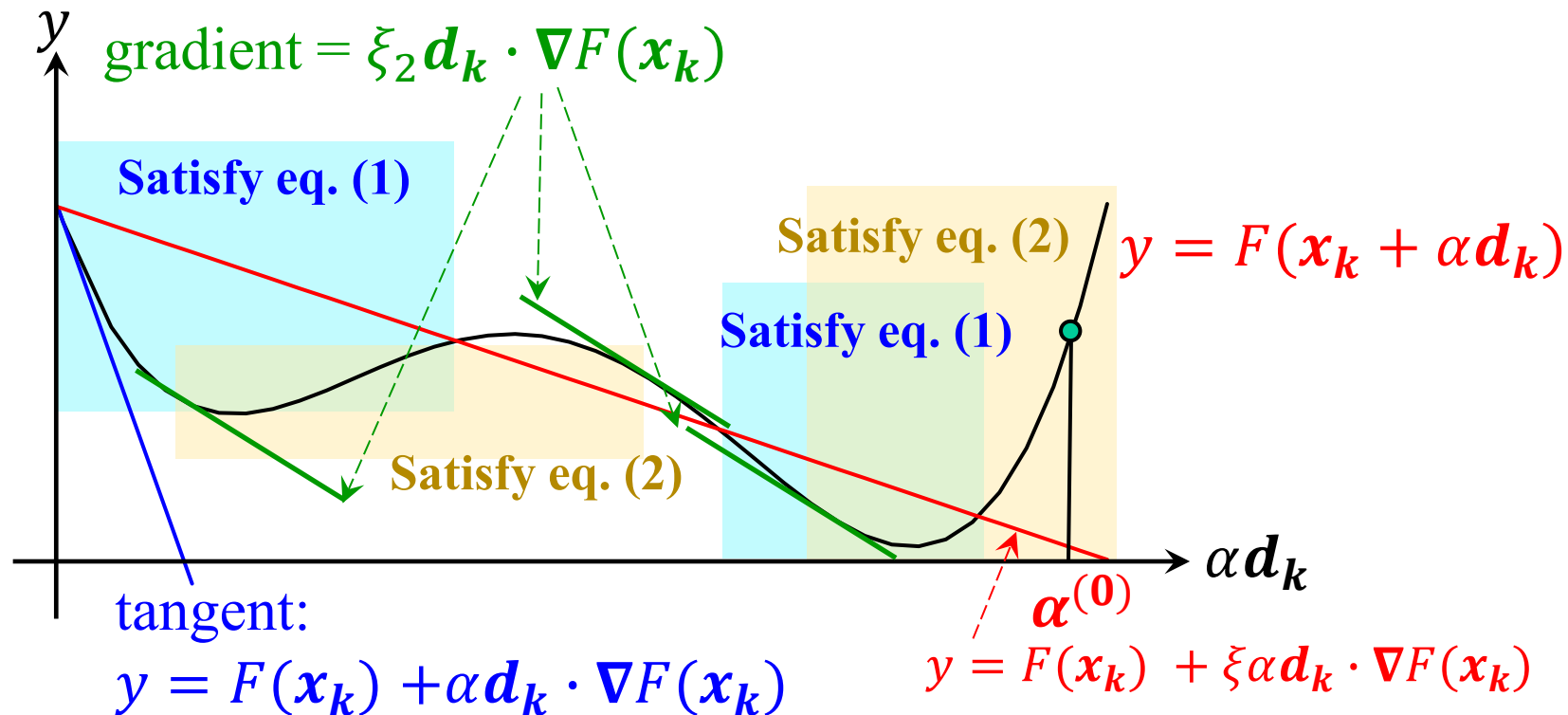
矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

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 $f(x)$

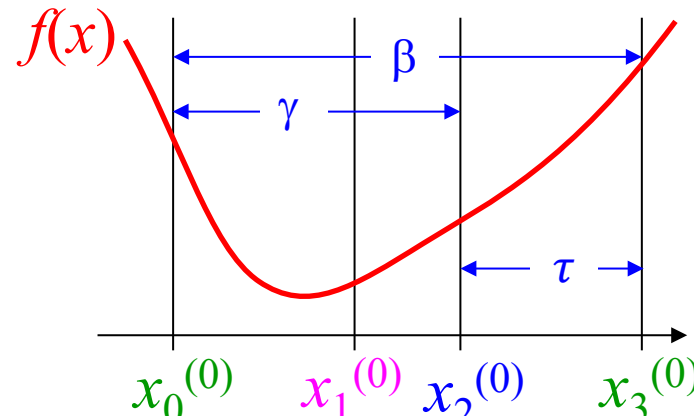
Unique solution exists in the range $[x_0^{(0)}, x_3^{(0)}]$ if $f(x_1^{(0)}) < f(x_0^{(0)}), f(x_3^{(0)})$ for $x_0^{(0)} < x_1^{(0)} < x_3^{(0)}$

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 $x_0^{(1)} := x_0^{(0)} < x_1^{(1)} := x_4^{(0)} < x_2^{(1)} := x_1^{(0)} < x_3^{(1)} := x_3^{(0)}$



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 $[x_0^{(0)}, x_2^{(0)}]$

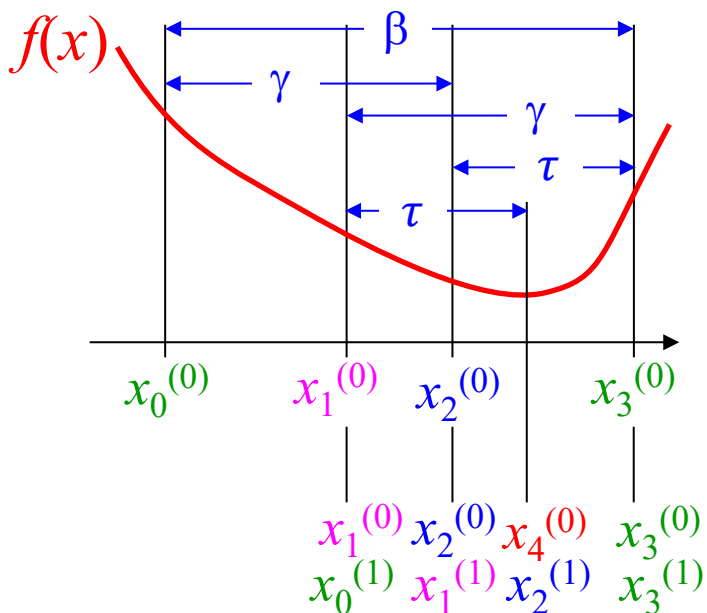
Replace $x_2^{(0)}$ with $x_4^{(0)}$ in $[x_0^{(0)}, x_1^{(0)}]$

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 $x_0^{(1)} := \mathbf{x}_1^{(0)} < x_1^{(1)} := \mathbf{x}_2^{(0)} < x_2^{(1)} := \mathbf{x}_4^{(0)} < x_3^{(1)} := \mathbf{x}_3^{(0)}$



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

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

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

$$\tau = \bar{\beta} - \gamma$$

$$\mathbf{x}_4^{(k)} = \mathbf{x}_1^{(k)} + \tau$$

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

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

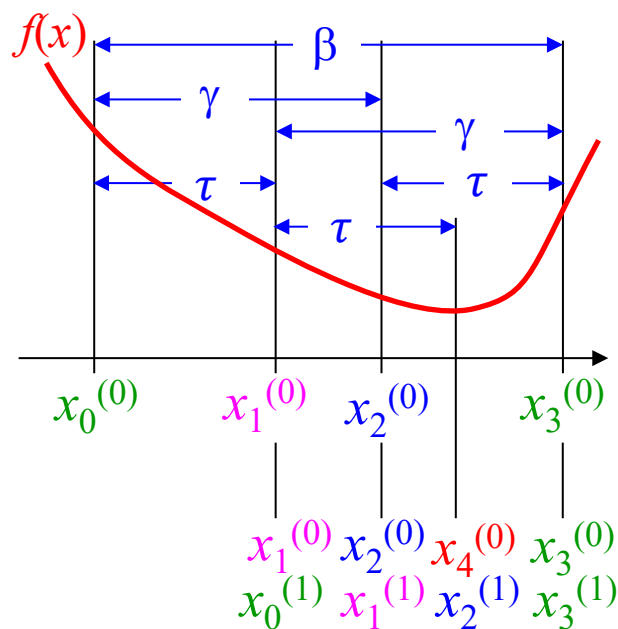
$$= x_3^{(k)} - x_1^{(k)} : x_4^{(k)} - x_1^{(k)} = \gamma : \tau$$

$\tau = \beta - \gamma$ 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. $\beta^{(k+1)} = \tau^{(k)}$

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

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

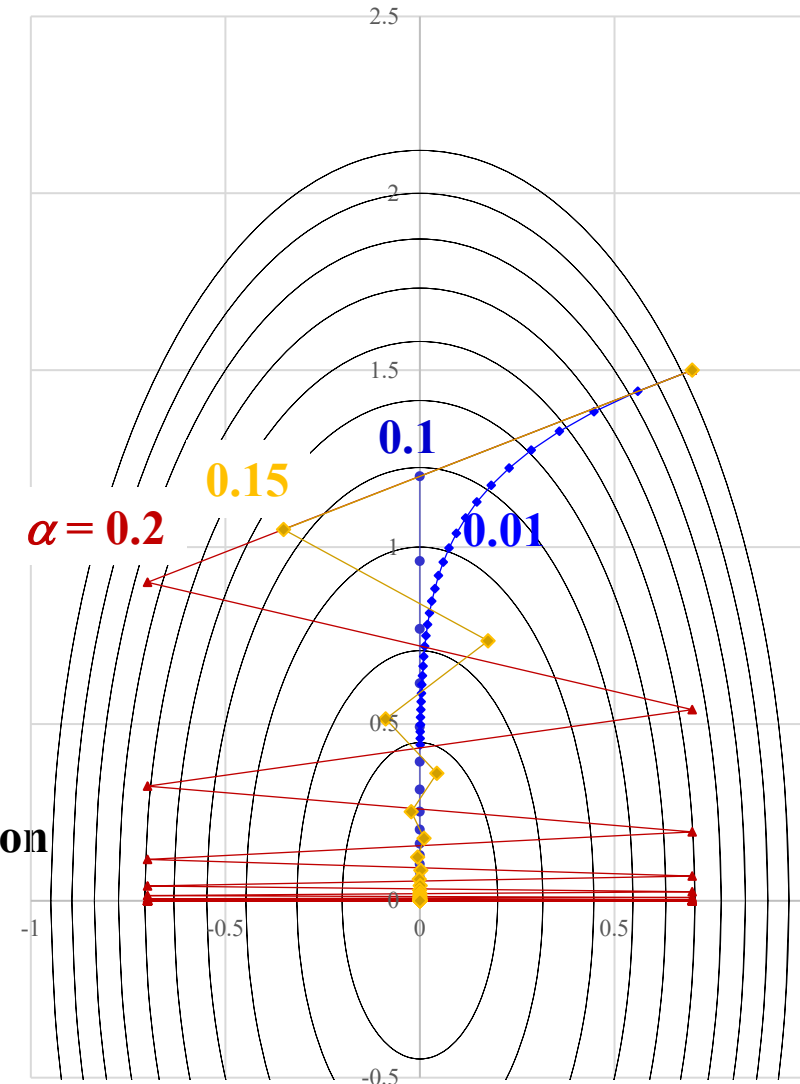
for quadratic problems

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

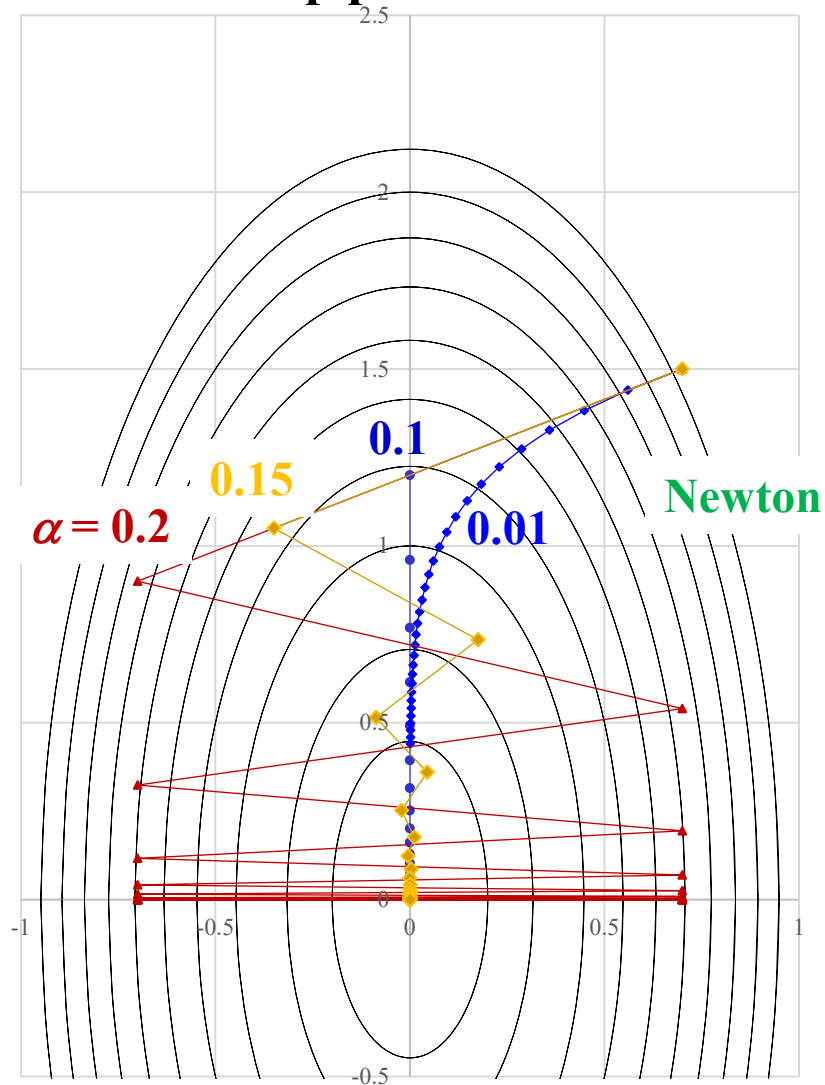
direction S^2 が大きく非対称な場合、最急勾配方向は最小値方向とは大きく異なることがある

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

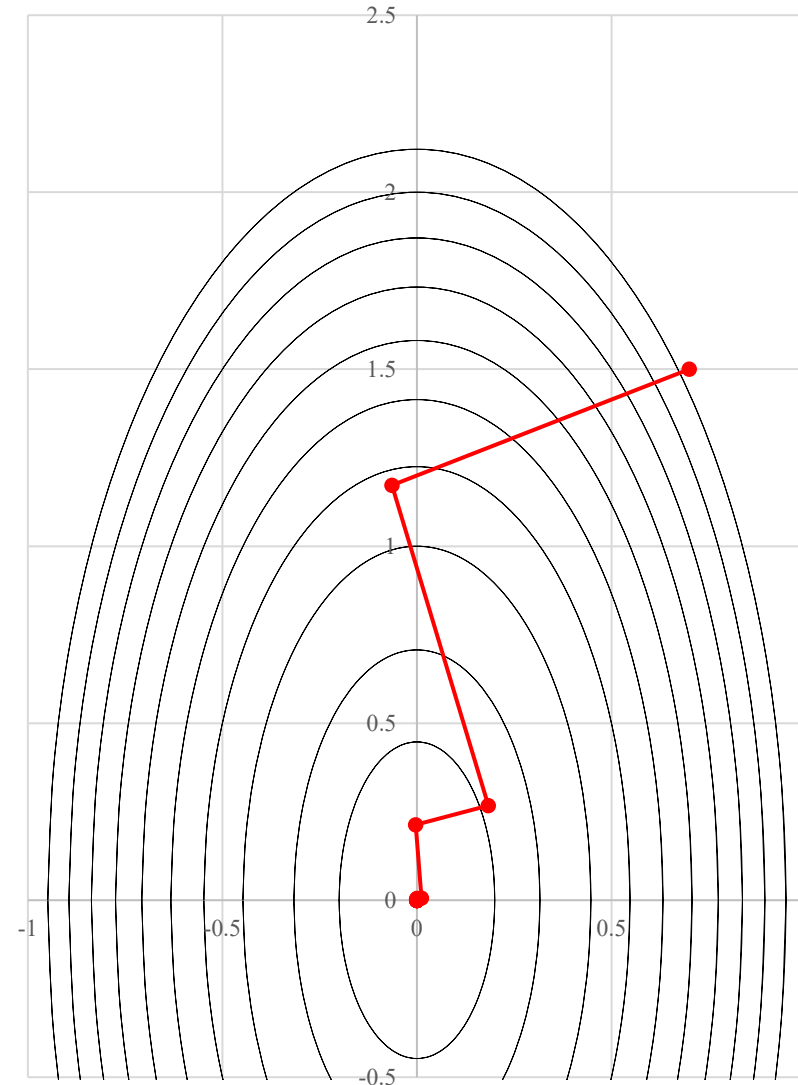


Steepest descend method

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:

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

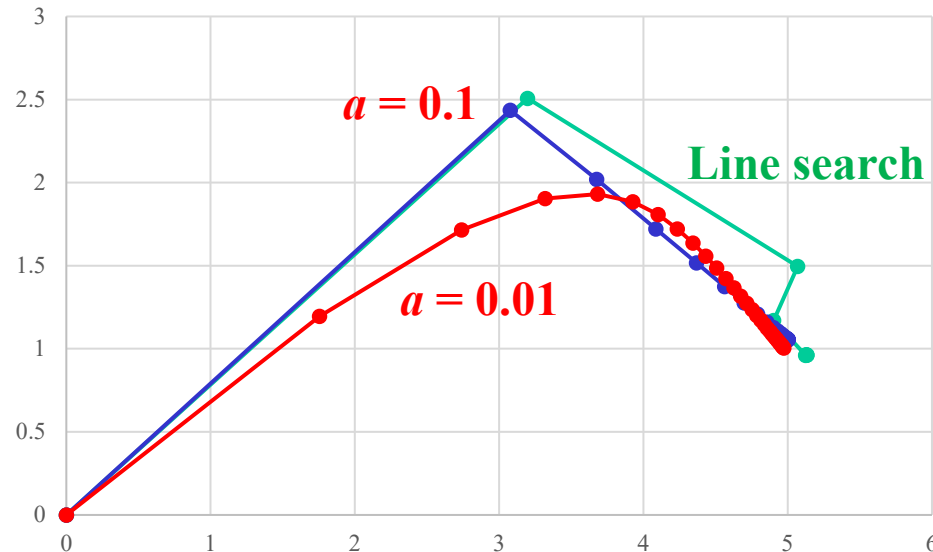
1000 batch data $(x_{1i}, x_{2i}, f(x_i))$ are generated at random

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

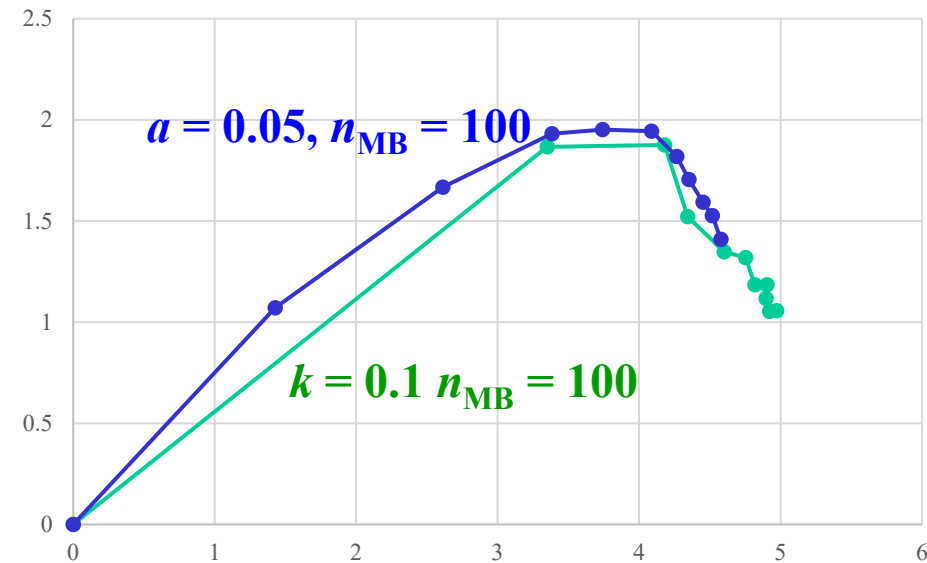
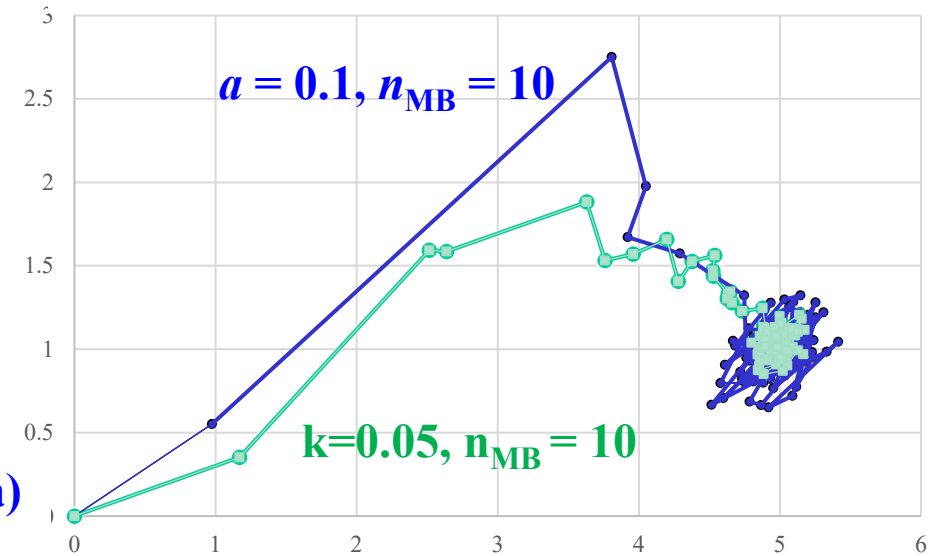
Speculate a and b

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

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

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

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

(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

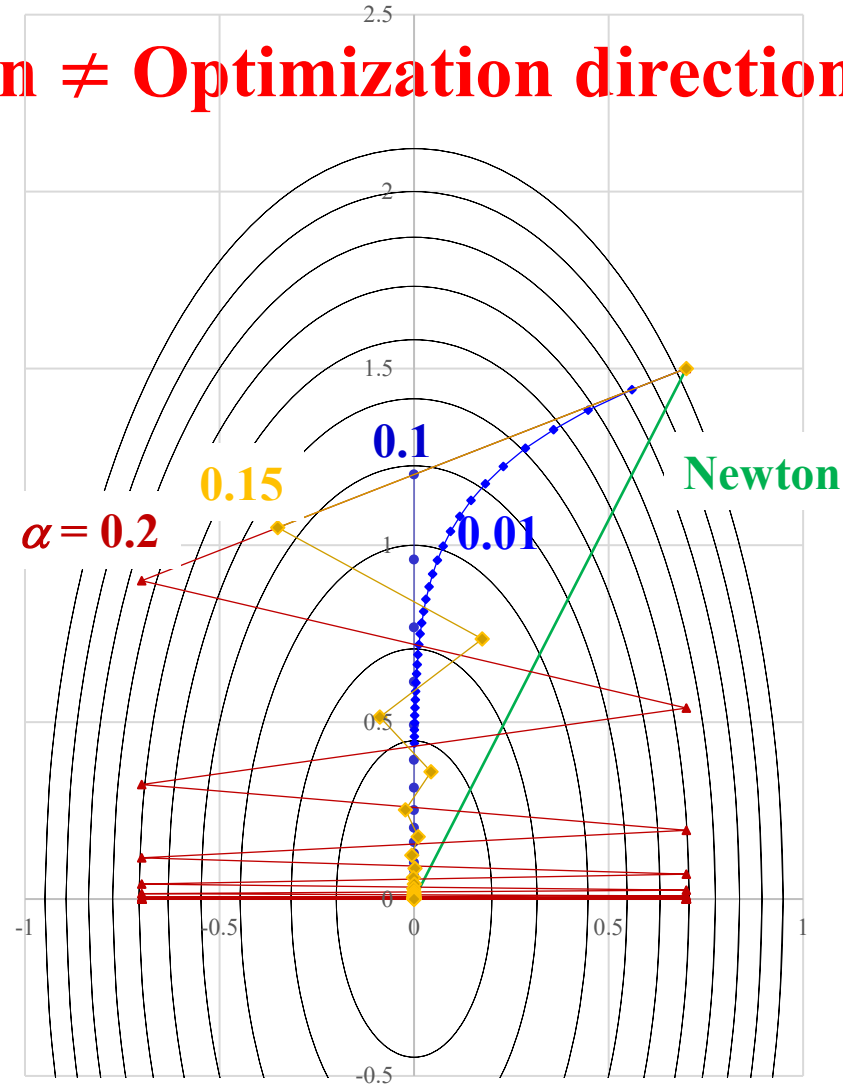
Convert F'' to positive definite and suppress divergence

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

λ : **Damping Factor**

SD vs. Newton-Raphson methods

Steepest direction \neq Optimization direction

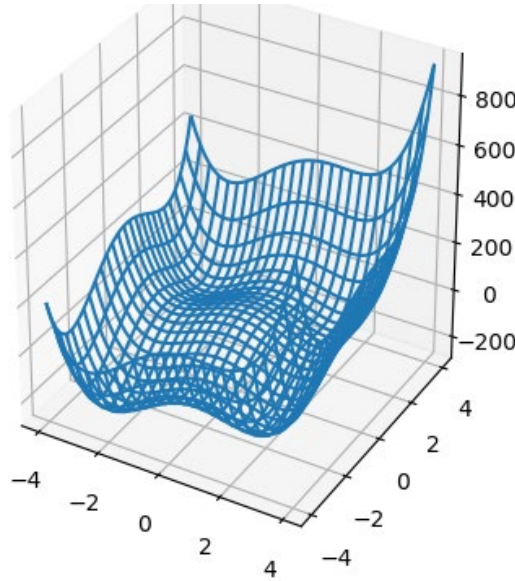


Improve SD method to follow optimization directions

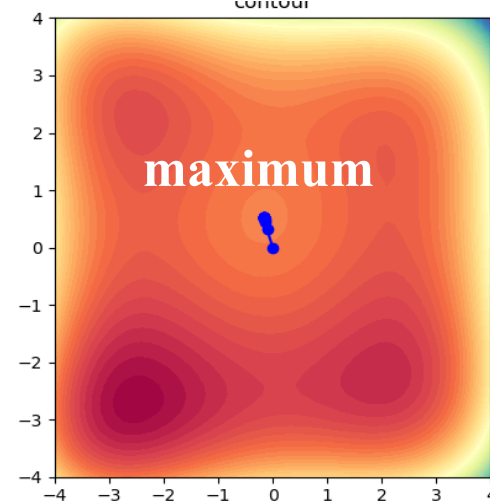
Program: optimize-newton-raphson2d.py

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

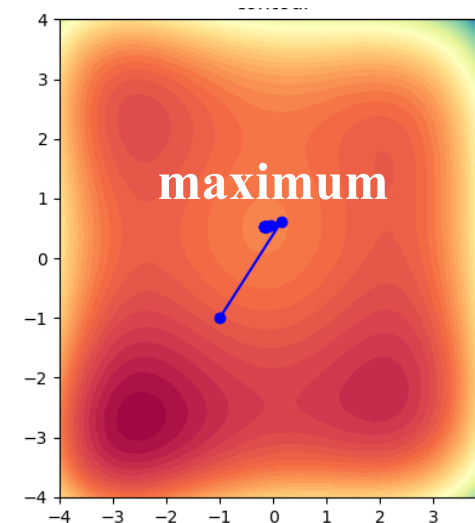
Usage: **python optimize-newton-raphson2d.py** x_0 y_0



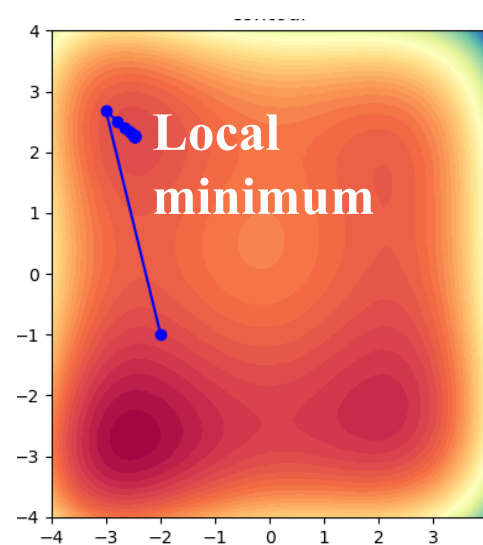
From (0.0 0.0) Newton



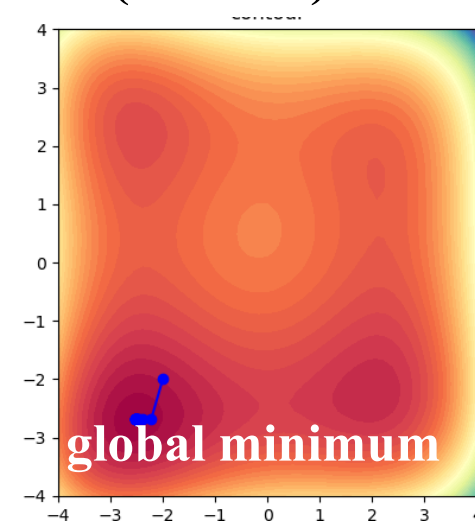
From (-1.0 -1.0)



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

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

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

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 achieved; or, go to STEP 5

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

Conjugate Gradient method (共役勾配法)

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

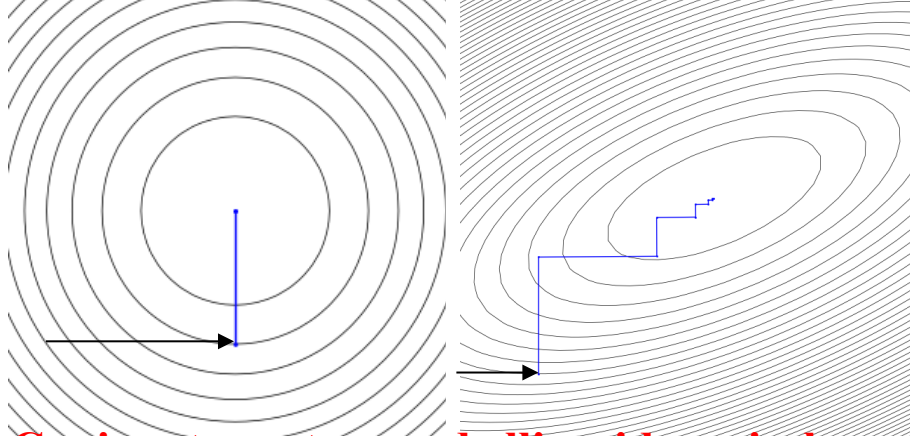
Vectors u and v satisfy $u^T A v = 0$ for a matrix A : u and v are conjugate with each other

- For quadratic function, repetition of the conjugate direction will find the minimum in finite cycles if exact line search is employed

共役な探索方向に沿って正確な直線探索を実行 \Rightarrow 有限回の反復で2次関数の最小解に到達

Case contour is a circle, one cycle calculation reaches the minimum

等高線が円の場合、一回の探索で最小値に到達できる



Conjugate vectors and ellipsoido – circle conversion

$$u^T P^T P v = u^T A v = 0$$

1. Give initial value x_0
2. Initial direction d is determined by SD
$$d = -\nabla f$$

3. Find x_{k+1} using appropriately chosen α_k
$$x_{k+1} = x_k + \alpha_k d_k$$

 α_k may be a small constant step or determined by a line search method

4. Search direction is updated by

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

$$d_{k+1} = -\nabla f(x_{k+1}) + \frac{\nabla f(x_{k+1})^T y_k}{d_k^T y_k} d_k$$

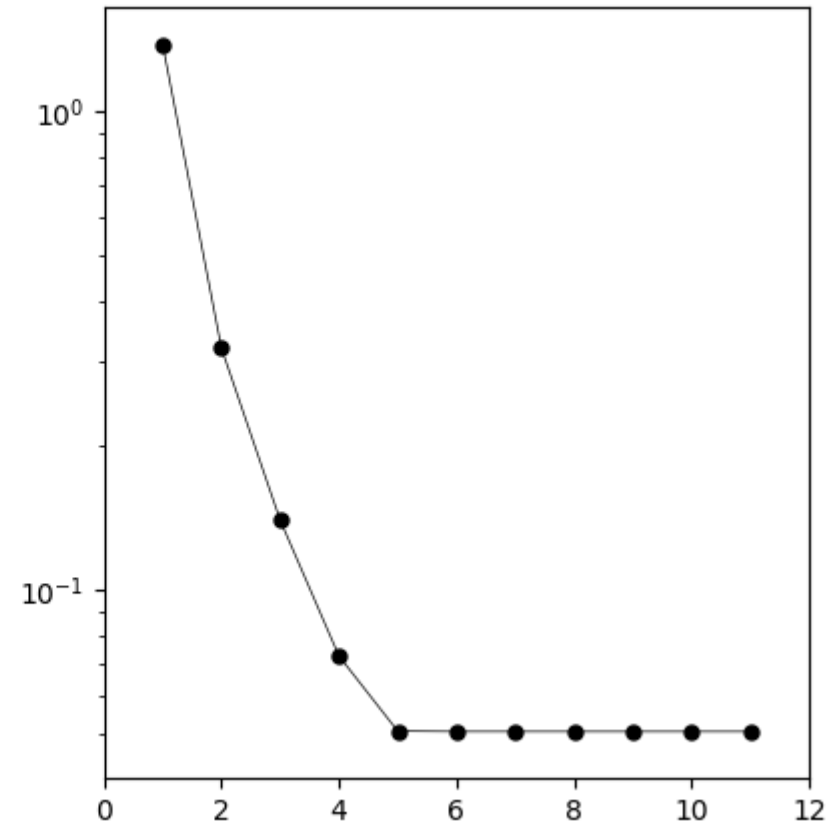
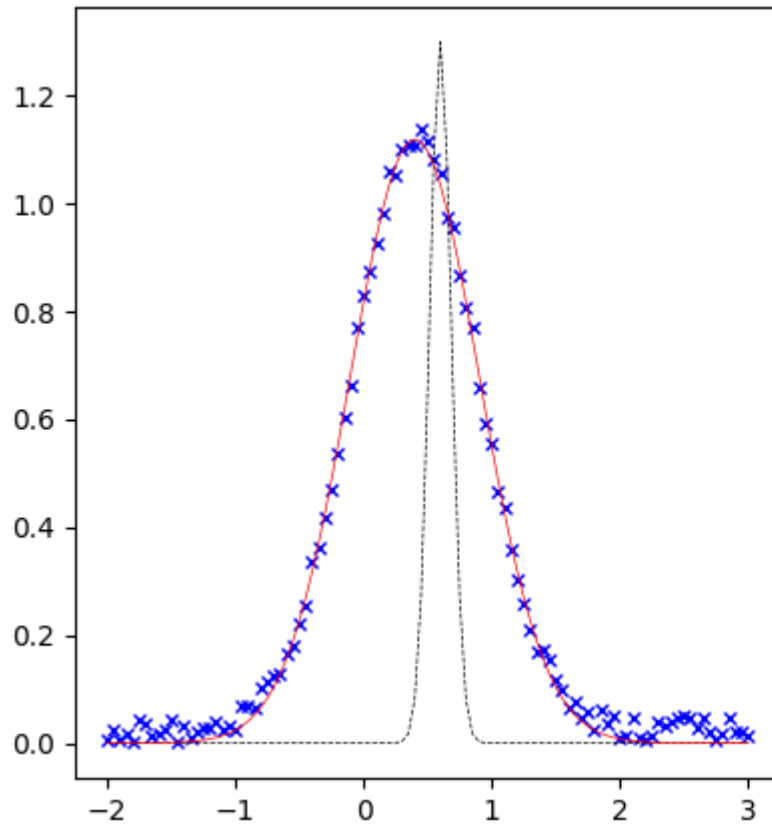
5. Repeat 3 – 4 to reach convergence

As the freedom of cg directions is the number of parameters (n_{param}), need to go back to 2 to reset d_k at some interval (typically n_{param} , necessary for $n_{\text{param}} = 2$).

Ex.: Curve fit

Usage: `python peakfit.py mode input_file method I0 x0 w`

python peakfit.py fit peak.xlsx (initial values: `method = cg`, `I0 = 1.3`, `x0 = 0.6`, `w = 0.1`)

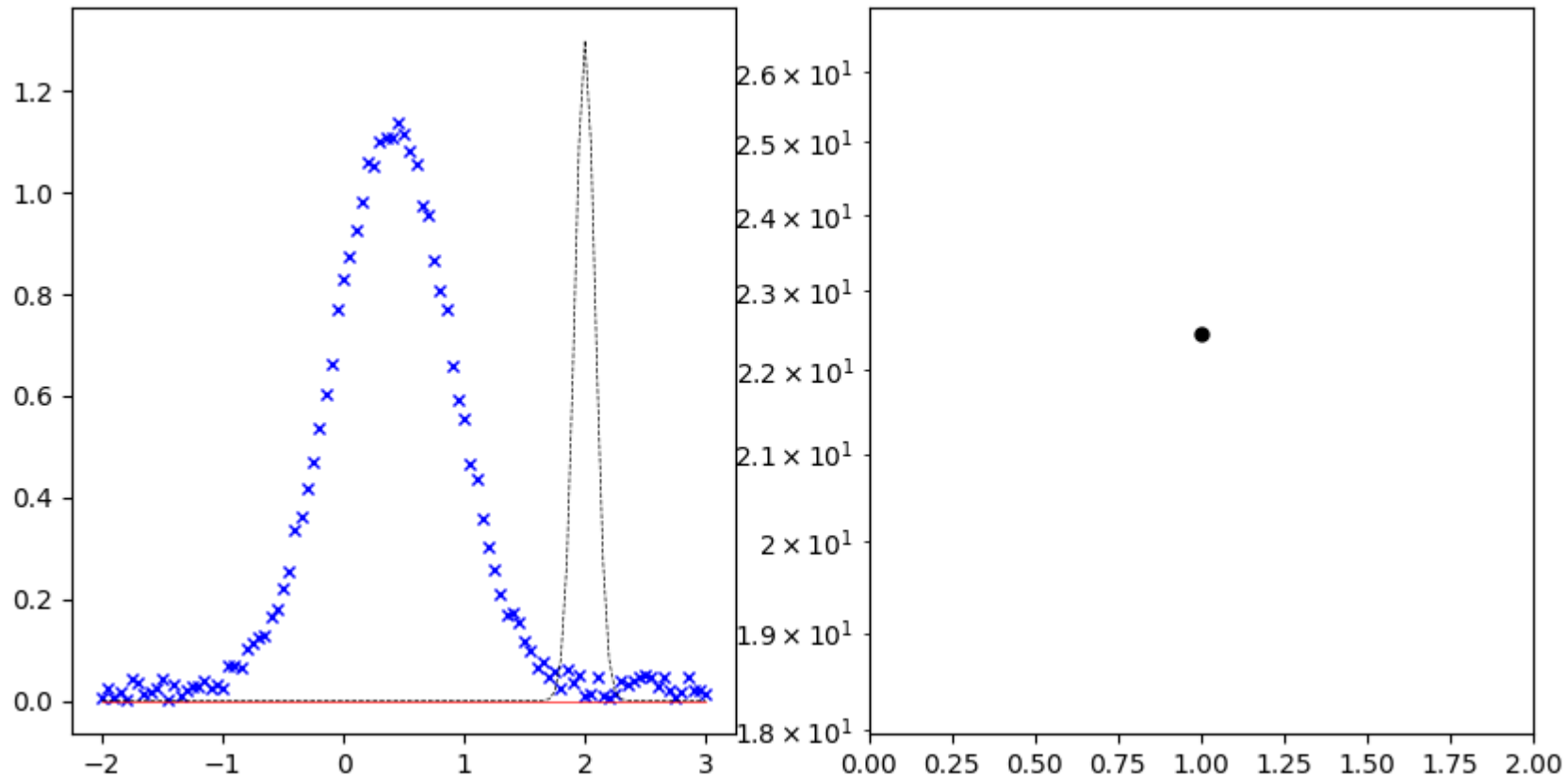


Not optimized

python peakfit.py fit peak.xlsx cg 1.3 2.0 0.1

method: cg

initial values: $I_0 = 1.3$, $x_0 = 2.0$, $w = 0.1$

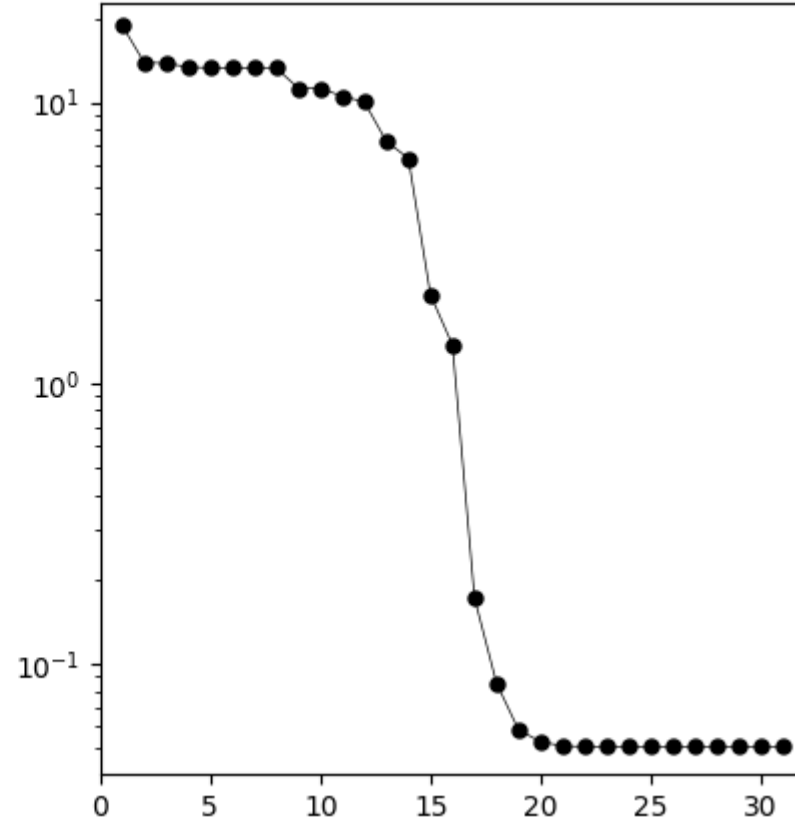
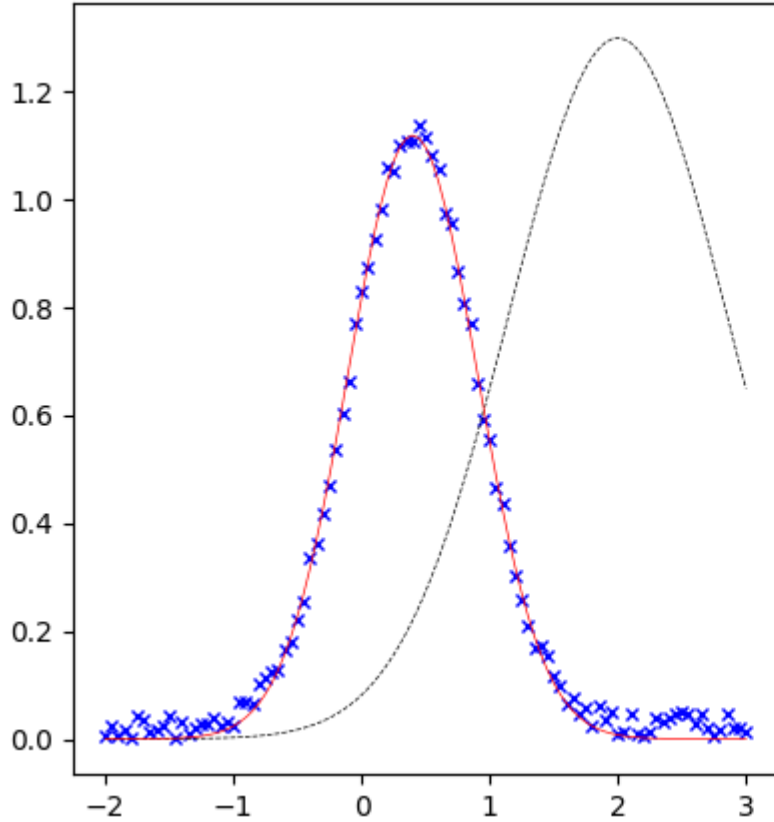


Converged

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

method: cg

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



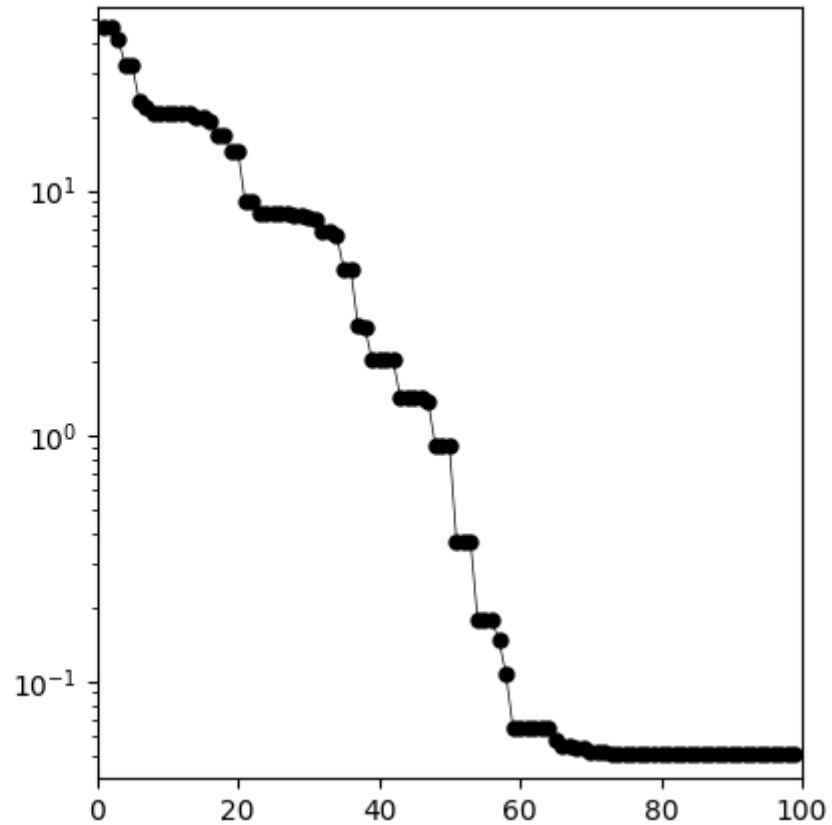
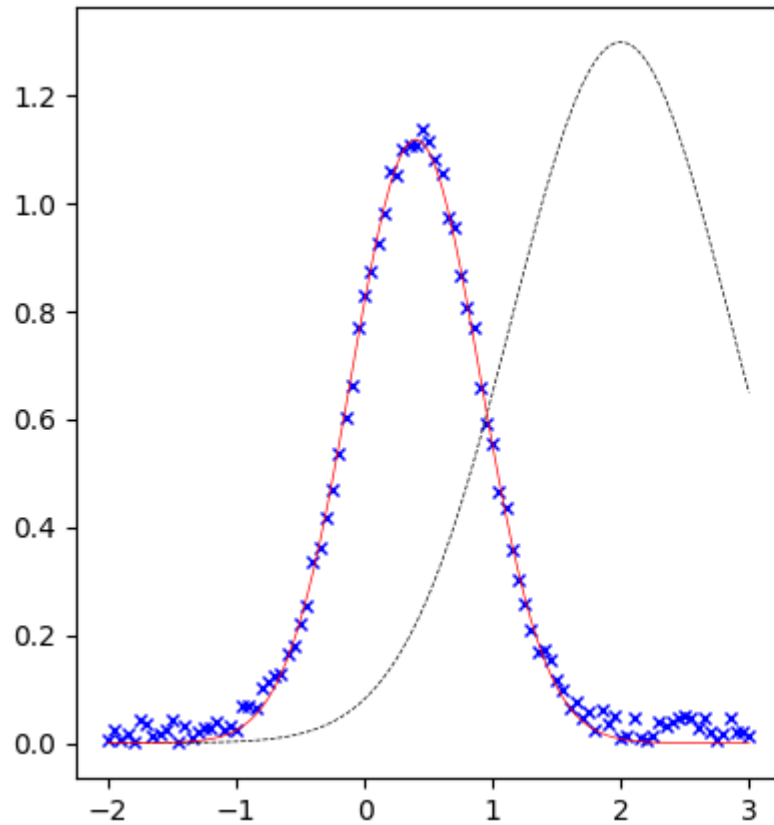
The result may be converged if w is wide enough to cover some region of the curve, even if the initial peak position is out of the FWHM of the target peak

Converged

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

method: nelder-mead (SIMPLEX)

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



Marquart method (マーカート法)

Minimize a square sum of m functions $f_j(x_i)$ with N parameters

$$F(x_i) = \sum_{j=1}^m f_j(x_i)^2$$

Approximate by

$$f_j(x_i + \delta x_i) \sim f_j(x_i) + \left(\frac{\partial f_j}{\partial x_k} \right) (\delta x_i) = f_j(x_i) + \mathbf{A} \delta x_i \quad A_{jk} = \frac{\partial f_j}{\partial x_k}$$

$$F(x_i + \delta x_i) \sim F(x_i) + 2 \sum_{j,k} f_j A_{jk} \delta x_k + \sum_{j,k,k'} A_{jk} A_{ik'} \delta x_k \delta x_{k'}$$

$$\frac{\partial F(x_i)}{\partial \delta x_k} \sim 2 \sum_j \left(A_{jk} f_j + \sum_k A_{ik} A_{jk} \delta x_j \right) = 0$$

$$\delta x = -(\mathbf{A}^t \mathbf{A})^{-1} \mathbf{A}^t (f_j) \quad \text{Gauss-Newton method}$$

Levenberg-Marquart method

$$\delta x = -(\mathbf{A}^t \mathbf{A} + \lambda I)^{-1} \mathbf{A}^t (f_j) \quad \lambda: \text{dumping factor}$$

$$\delta x = -(\mathbf{A}^t \mathbf{A} + \lambda \text{diag}(\mathbf{A}^t \mathbf{A}))^{-1} \mathbf{A}^t (f_j) \quad \text{e.g. chosen proportional to diagonal sum of } \mathbf{A}^t \mathbf{A}$$

Simplex method (単体法, Amoeba法)

(Nelder-Mead algorithm)

服部力、名取亮、小国力 監修、Fortranによる数値計算ソフトウェア、丸善株式会社 (1989年)

Simplex: Polyhedron formed by $(n+1)$ vertexes in n -dimension space

(単体: n 次元空間で $(n+1)$ 個の頂点で作る多面体)

Minimize $F(\mathbf{x}_i)$

1. $(n+1)$ initial values \mathbf{x}_i ($i = 1, 2, \dots, n+1$) \Rightarrow Sort $F(\mathbf{x}_i)$ so that $F(\mathbf{x}_i) > F(\mathbf{x}_{i'})$ ($i < i'$)

$$\mathbf{x}_h = \mathbf{x}_1, \mathbf{x}_1 = \mathbf{x}_{n+1}$$

2. Average except the maximum vertex \mathbf{x}_i $\mathbf{x}_G = \sum_{i=2}^{n+1} \mathbf{x}_i / n$

3. New x will be examined along the line $\mathbf{x}_1 - \mathbf{x}_G$ by the following selections

(i) Reflection (鏡映) : $\mathbf{x}_R = (1 + \alpha)\mathbf{x}_G - \alpha\mathbf{x}_1$ ($\alpha > 0$, ex. 1.0)

(ii) Expansion (拡大) : $\mathbf{x}_E = \gamma\mathbf{x}_R + (1 - \gamma)\mathbf{x}_G$ ($\gamma > 0$, ex. 2.0)

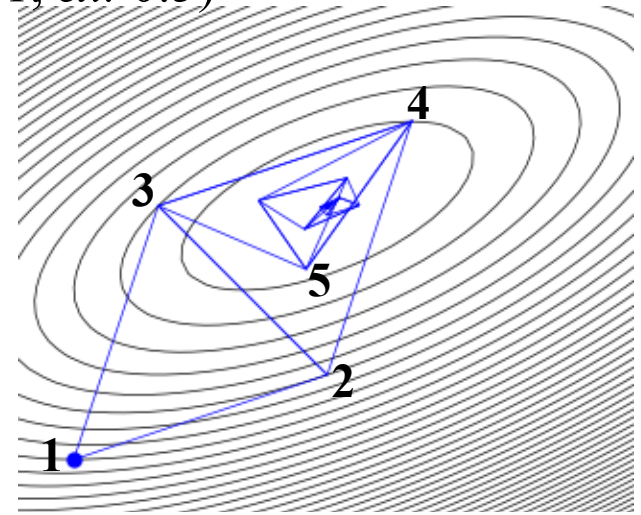
(iii) Contraction (収縮) : $\mathbf{x}_C = \beta\mathbf{x}_1 + (1 - \beta)\mathbf{x}_G$ ($0 < \beta < 1$, ex. 0.5)

(iv) Reduction (縮小) : $\mathbf{x}_{RD} = (\mathbf{x}_1 + \mathbf{x}_i) / 2$

4. Replace \mathbf{x}_1 with the x in (i) – (iv) that firstly satisfies

$$F(\mathbf{x}) < F(\mathbf{x}_1)$$

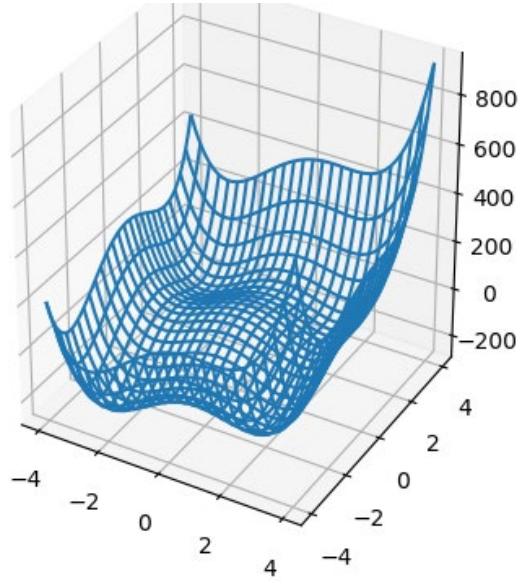
5. Repeat 2 - 4



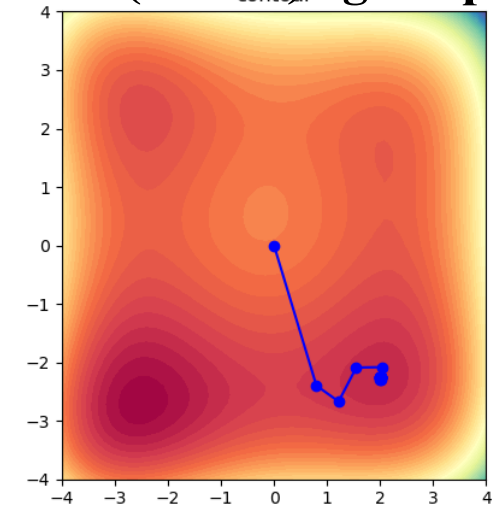
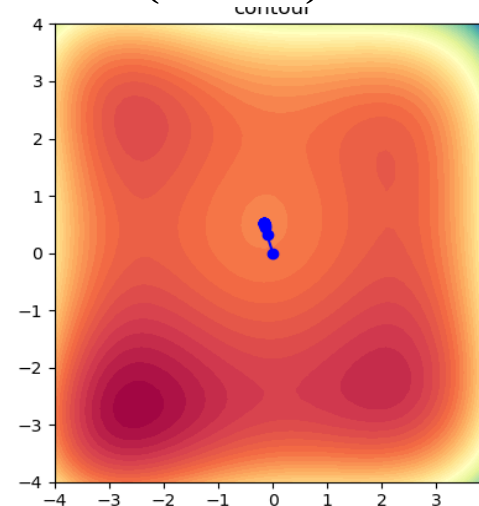
Comparison

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

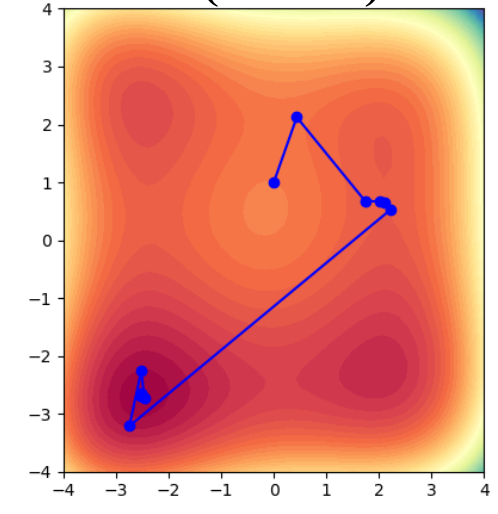
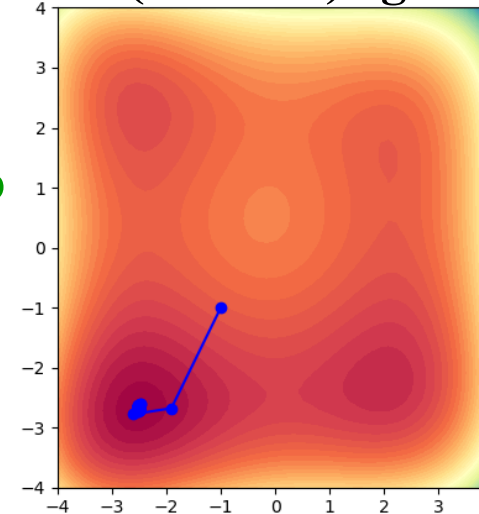
Programs: `optimize-sd-cg2d-linesearch.py`, `optimize-newton-raphson2d.py`



From (0.0 0.0) Newton From (0.0 0.0) cg simple



From (-1.0 -1.0) cg simple From (0.0 1.0) SD armijo

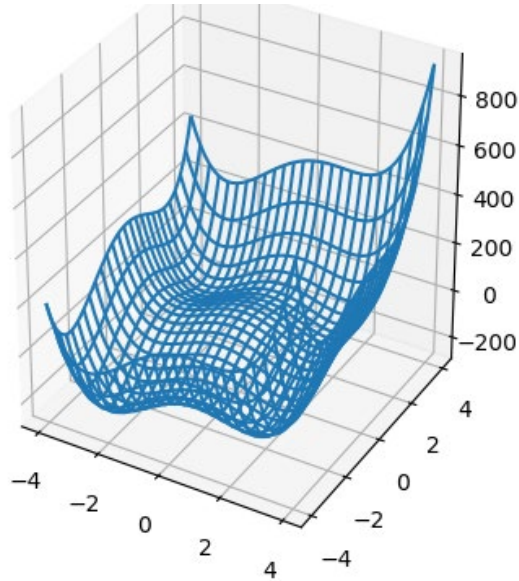


e.g.,
`python optimize-sd-cg2d-linesearch.py -1.0 -1.0 sd armijo`

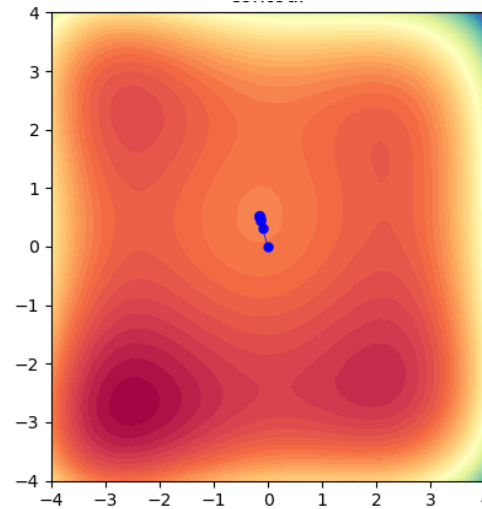
Optimize.py for Simplex method (not working without tklib)

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

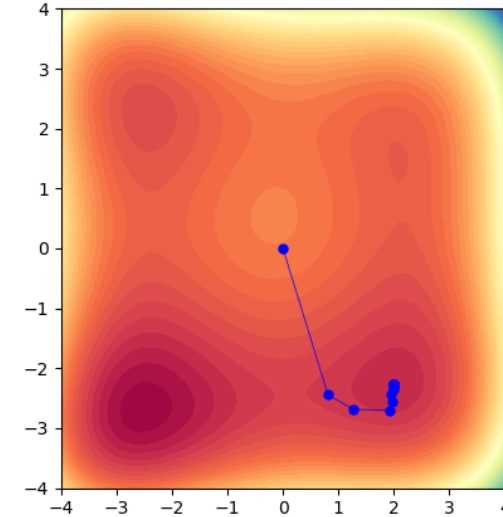
e.g.: python optimize.py 0.0 1.0 simplex



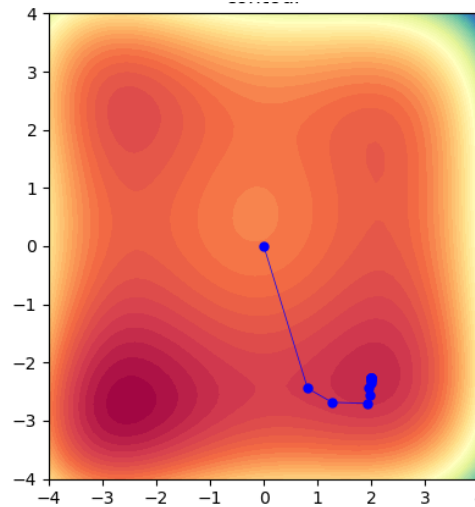
From (0.0 0.0) Newton



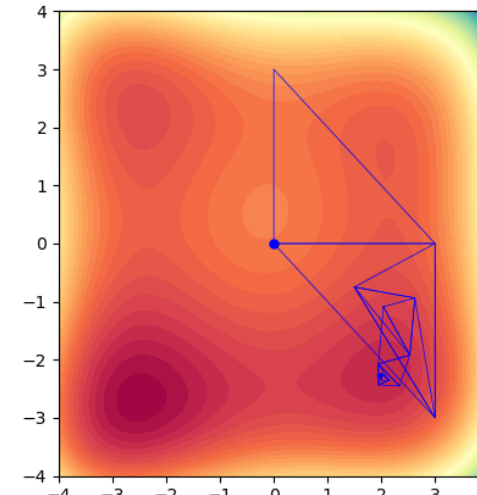
From (-1.0 -1.0) DFP golden



From (0.0 0.0) BFGS golden



From (0.0 1.0) Simplex



Main algorism:

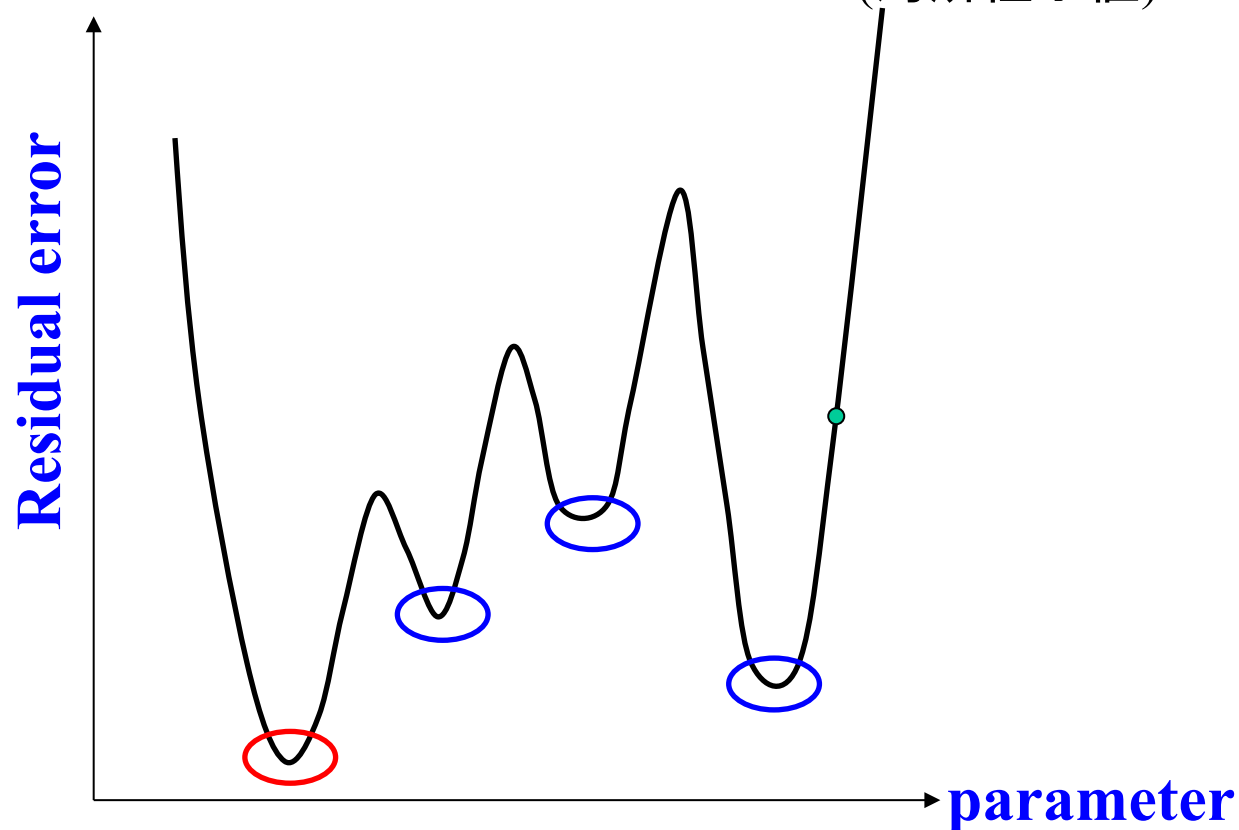
newton, sd, cg, broyden, dfp, bfgs
simplex

Direct search:

exact, one, simple
armijo, golden

Notes for NL optimization

- Solutions may be more than one
- Final solution is not obtained by one step calculation
- **Convergence must be confirmed**
- **Confirm the solution is the global minimum** (大域最小値)
 - ⇔ **Often fall in a local minimum** (局所極小値)



Features of NL optimization algorithms

Convergence	A	B
Speed	×	○
Stability	○	×
Global convergence	○	×
For:	Initial cycles	Later cycles for fast convergence

A : Simplex (単体法)

A,B: with line search algorithm:

Conjugate Gradient (CG, 共役勾配法)

Steepest Descent (SD, 最急降下法)

Quasi Newton methods

▪ **Davidson-Fletcher-Powell (DFP)**

▪ **Broyden-Fletcher-Goldfarb-Shanno (BFGS)**

B : Newton-Raphson method

Summary of non-linear (NL) optimization

Gradient method (勾配法): Use first differential to find the direction of minimum

- **Steepest Descent method (最急降下法):**
Only 1st differentials are used to search minimum
Simple program, Slower convergence than NR and CG
- **Newton-Raphson method: Gradient method**
Use 1st and 2nd differentials (Hessian matrix) to efficiently find minimum
Fast convergence, easily diverged, complex program
May reach to a maximum if Hessian matrix is not positive definite.
- **Marquart method**
For least-squares fitting of $f_j(x_i)$, 2nd differential matrix is build from 1st differentials of $f_j(x_i)$
Simple program, Slower convergence than NR

Summary of non-linear (NL) optimization

Gradient method (勾配法): Use first differential to find the direction of minimum

- **Quasi Newton method (準Newton法): DFP, BFGS, Broyden etc**
2nd differential matrix (Hessian matrix) is iteratively approximated from 1st differentials.
Line search method is combined to improve global convergence.
Better convergence by combining with linear search algorithms.
- **Conjugate Gradient method (共役勾配法):**
Search direction is corrected by conjugate gradient of 1st differentials for efficient search
Better convergence than NR, faster than SD, complex program

Direct search method (直接探索法)

- **Simplex method (単体法)**
Trial and error with a pre-defined procedure to select next candidate parameters
Very slow but good convergence