

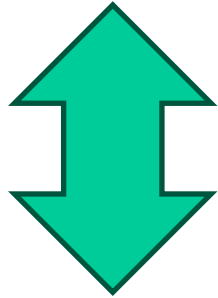
**How to solve equations?**

**Self-consistent method**

自己無撞着法

## Problems whose final unique solution is obtained just by one step calculation

- Linear least-squares method
- Up to 4<sup>th</sup> order polynomial equations



## 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)

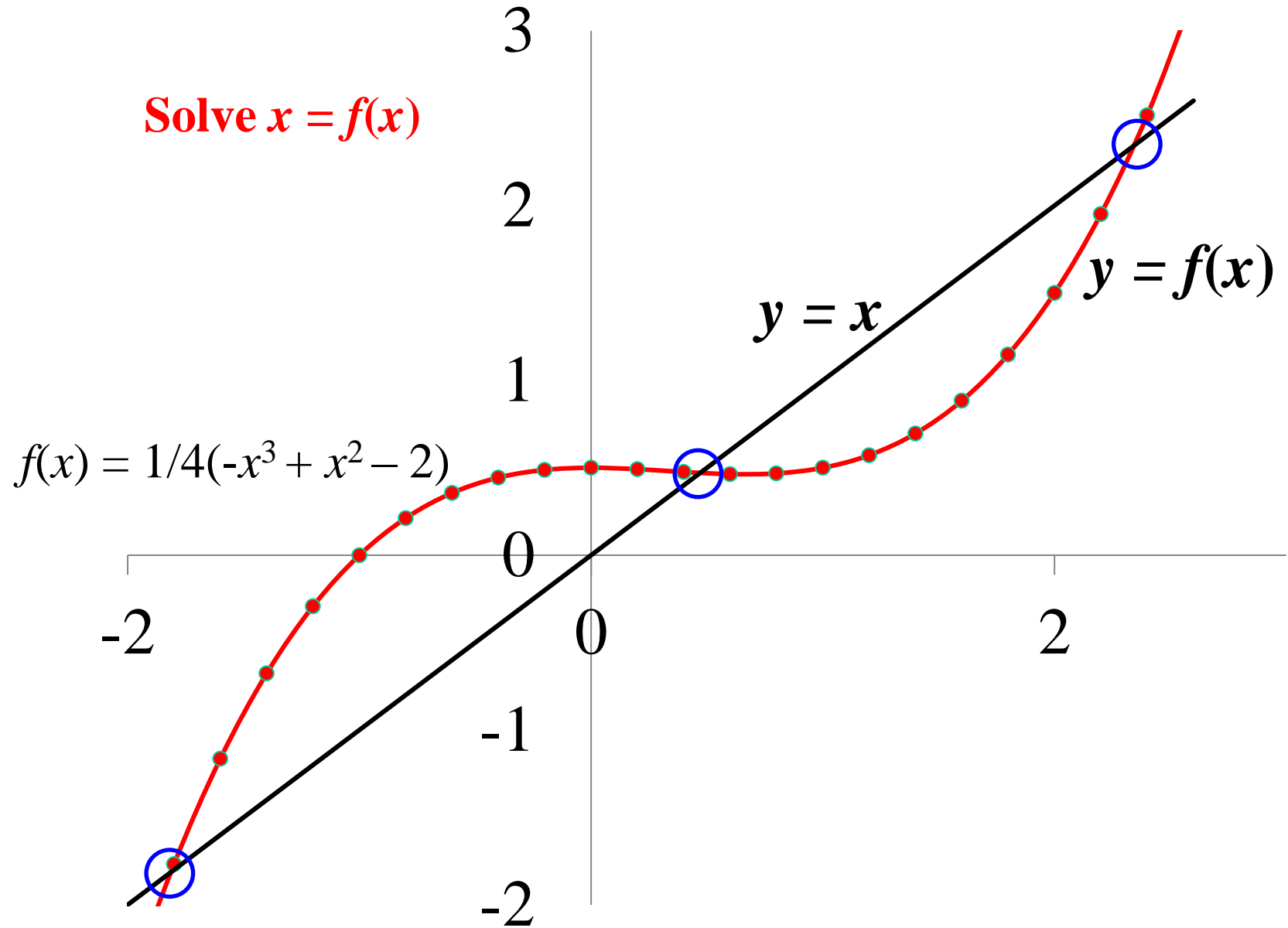
**Mixing factor (混合係数)  $k_{\text{mix}}$ : Stabilize convergence**

**Initial value  $x_0$**

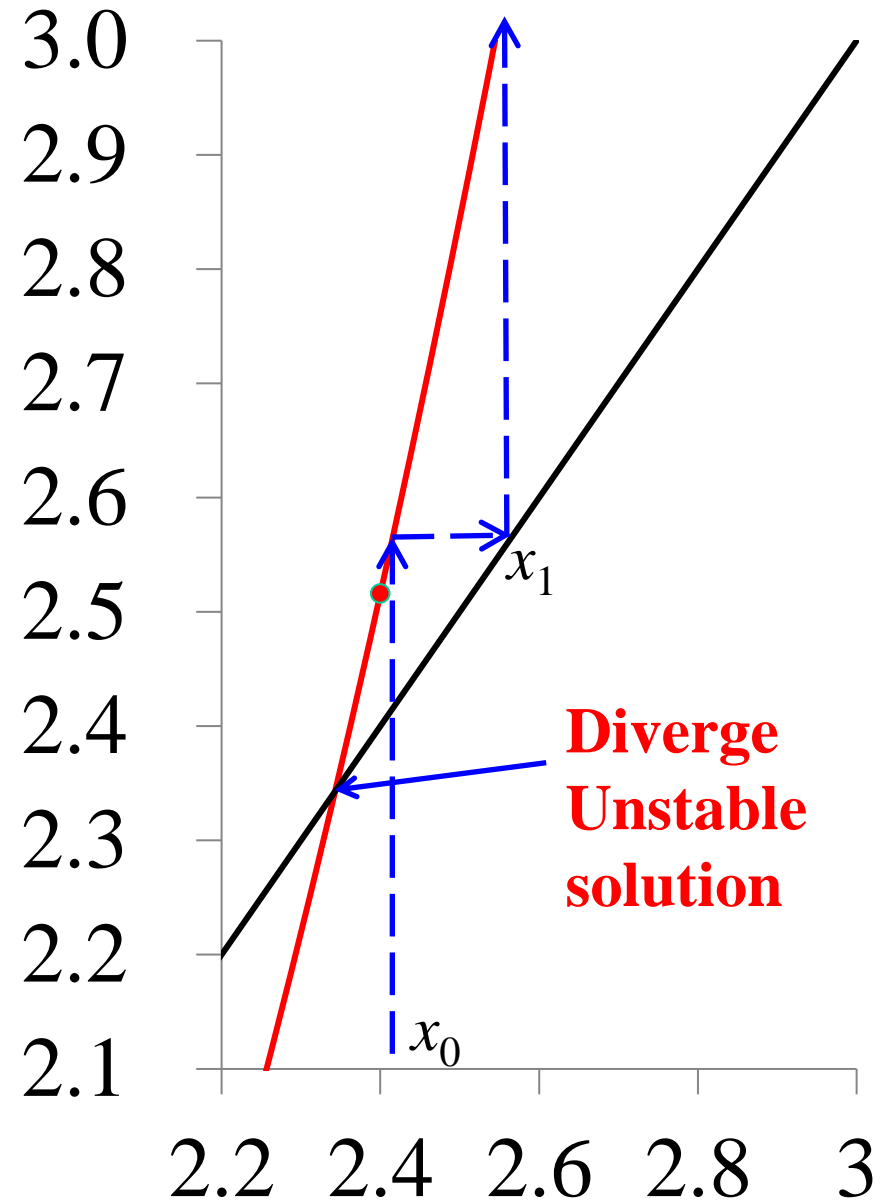
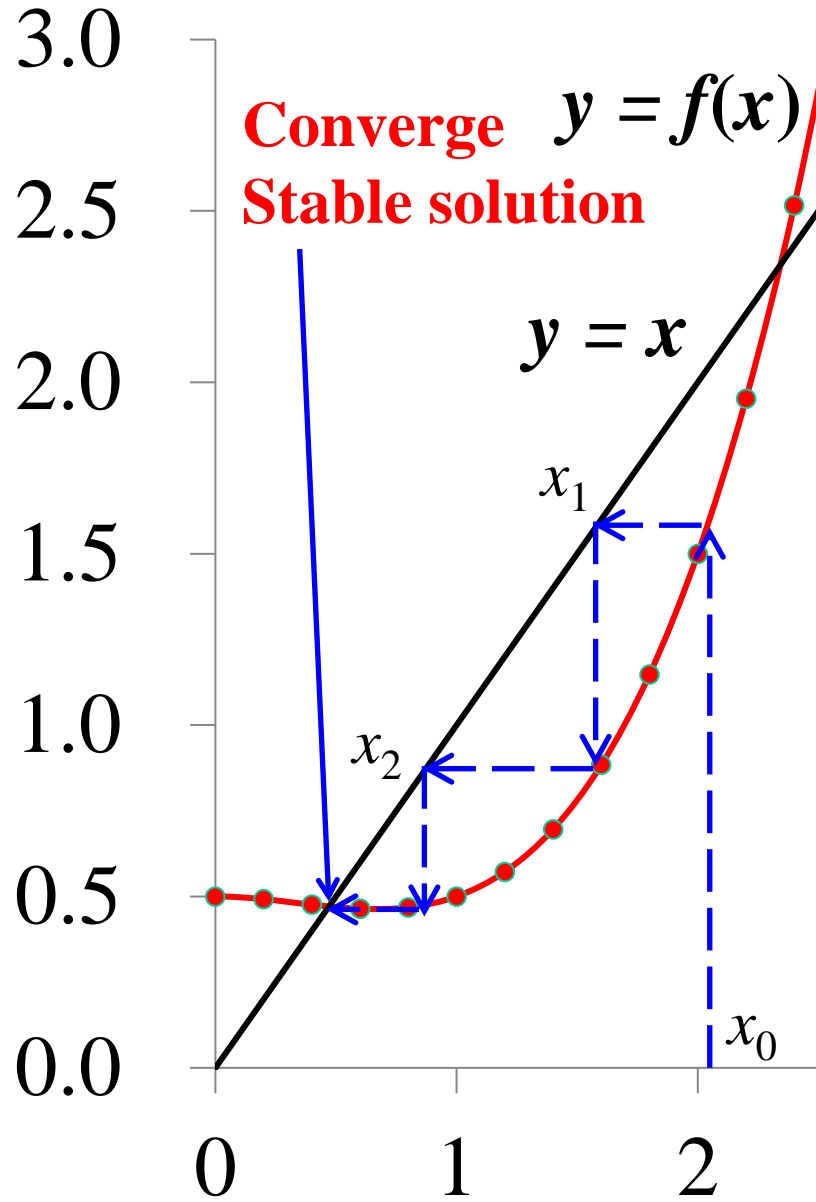
**1st iteration :  $x_1 = f(x_0) \Rightarrow x_1' = (1 - k_{\text{mix}}) x_0 + k_{\text{mix}} x_1$**

**2nd iteration:  $x_2 = f(x_1')$  ....**

# 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  $\rho_{ini}$ :  
e.g., by uniform density, sum of atomic electron density,,



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



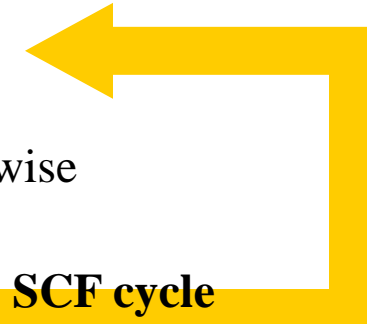
$\rho_{ini}$  must be equal to  $\rho_{fin}$ , otherwise these loss physical meaning

- More appropriate  $\rho_{new}$  is guessed from  $\rho_{fin}$  and  $\rho_{ini}$ , and repeat the above calculations

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

$k_{mix}$  : **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  $\rho_{fin} = \rho_{ini}$

# 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 convergence using mixing factor** (混合係数)  $k_{\text{mix}}$

Initial  $x_0$

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

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

# 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

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

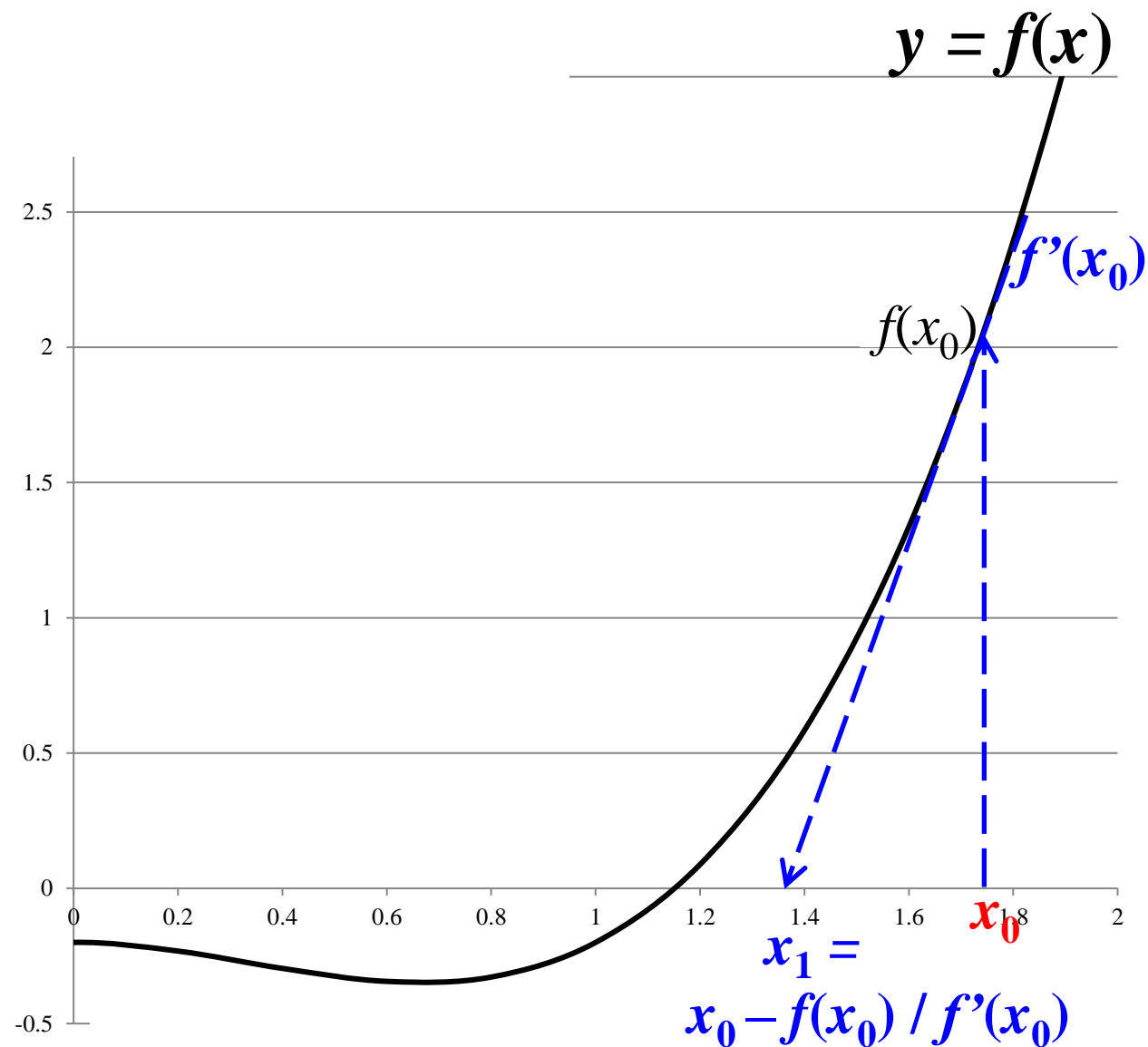
$$\Rightarrow x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$$

Stabilize convergence:

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

e.g.,  $\alpha_k = 1 / (1 + \lambda)$

$\lambda$ : Damping Factor



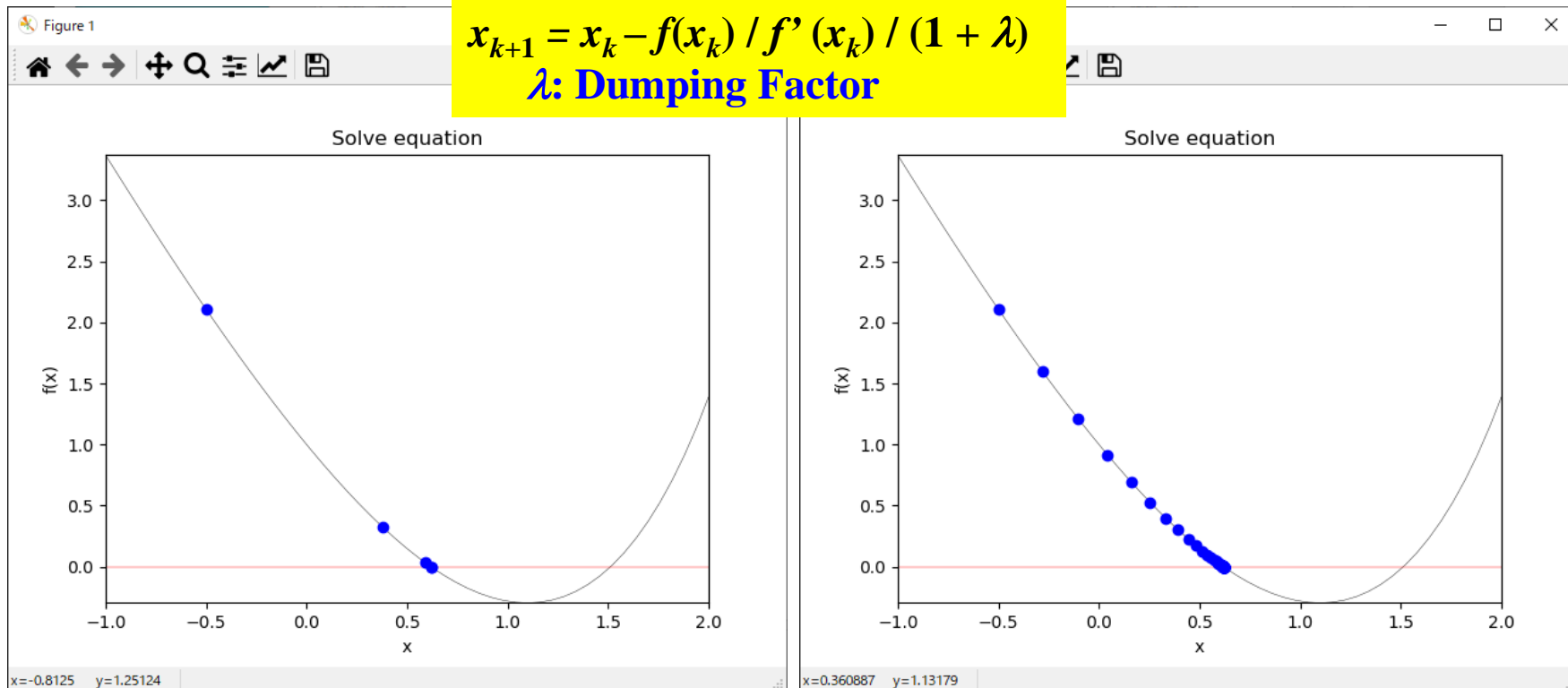
# Program: equation-newton-Raphson.py

Usage: python equation-newton-raphson.py x0 dump t<sub>sleep</sub>

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

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:  $x_0$

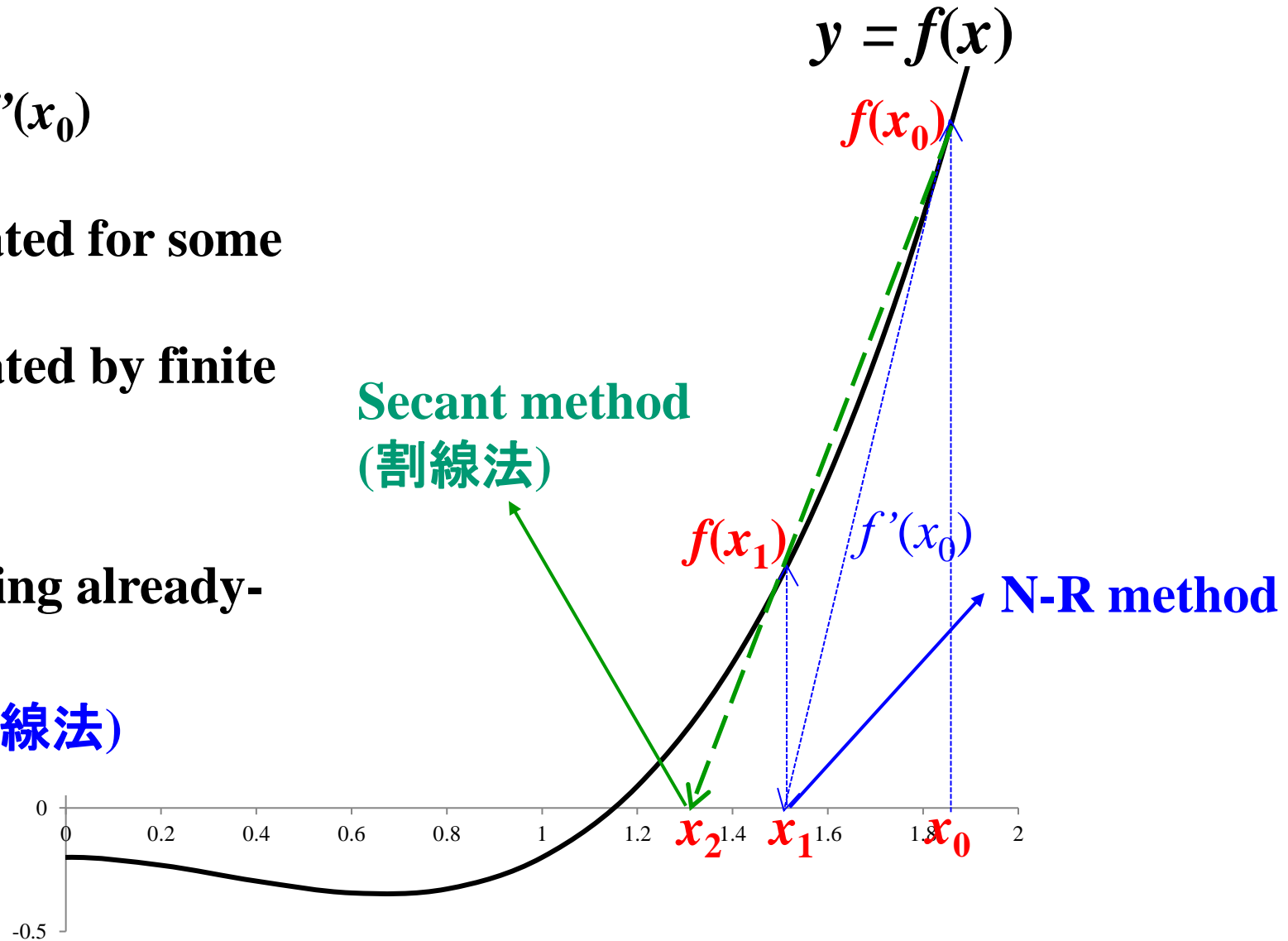
repeat:  $x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$

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

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

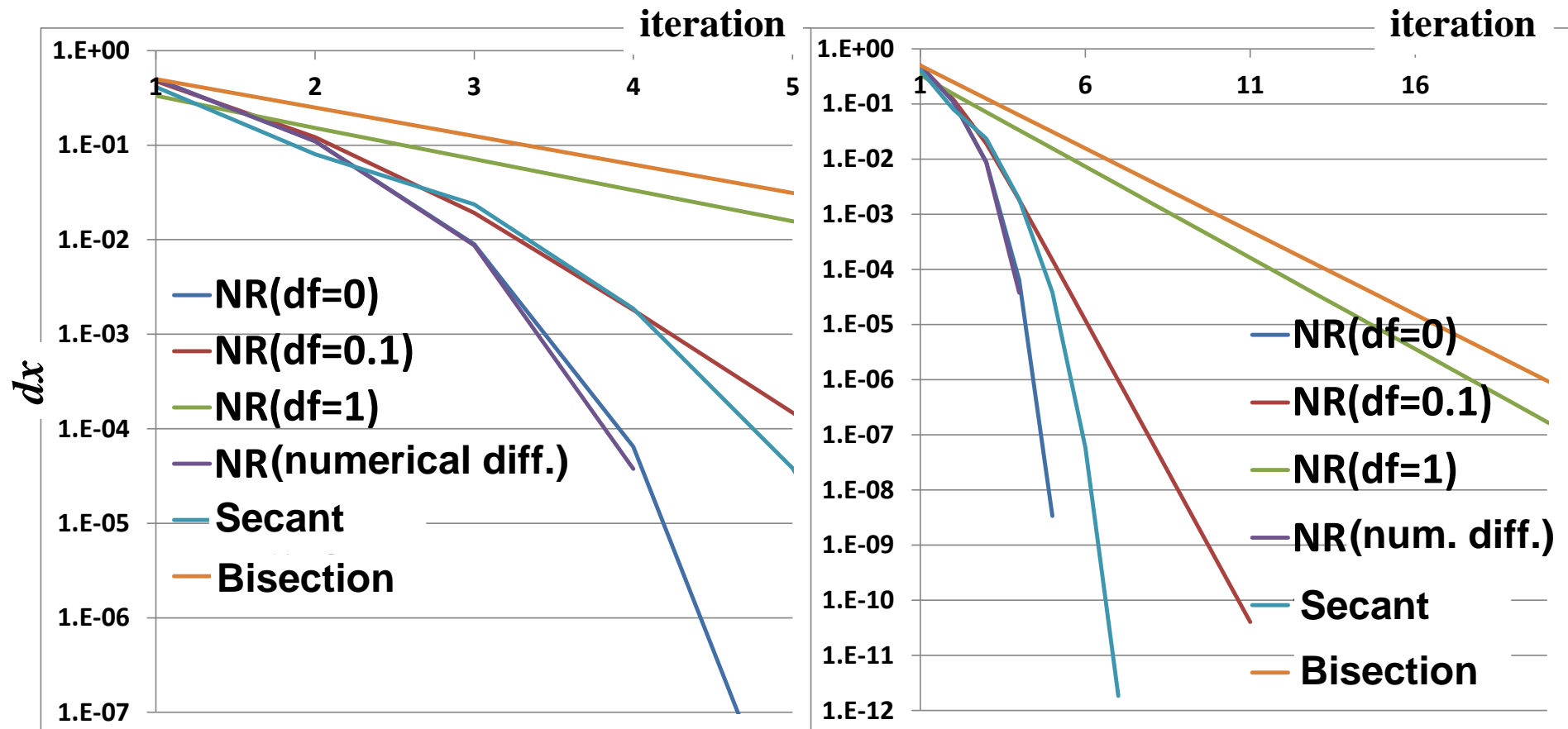
- $f'(x_0)$  can be approximated using already-calculated values

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



# Effect of dumping factor: Convergence process

$f(x) = \exp(x) - 3x = 0$  (initial  $x = 0$ )      An exact root: 0.619061



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

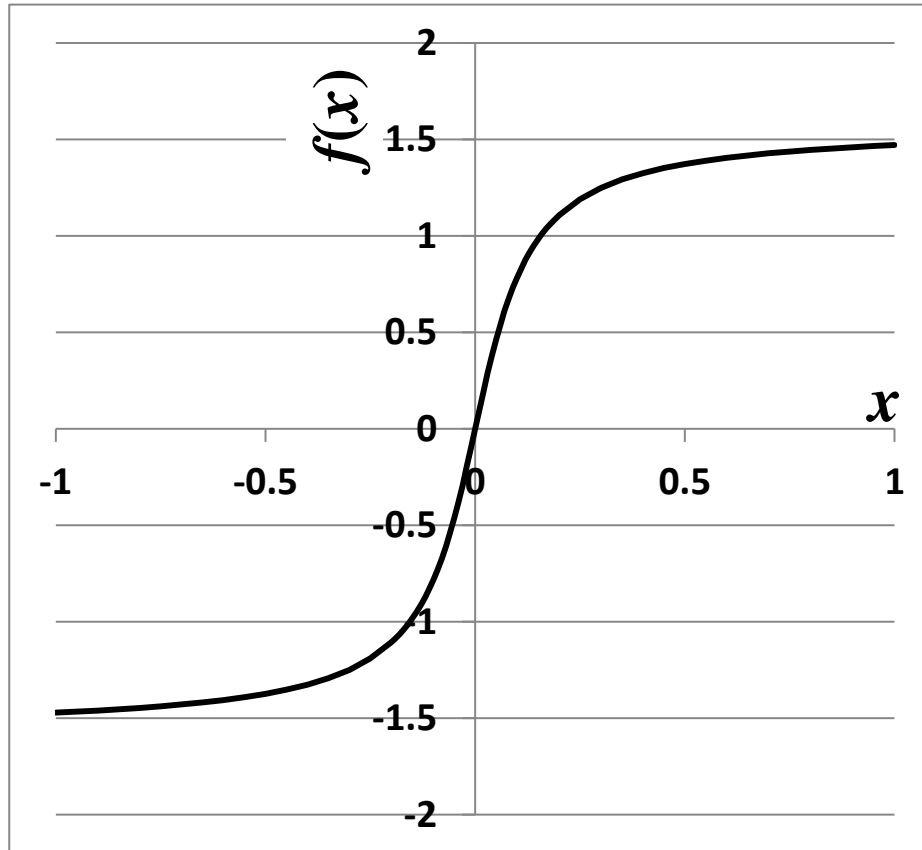
$\lambda$ : Dumping Factor

NR: Newton-Raphson method  
df: Dumping Factor

# Case Newton method succeeds

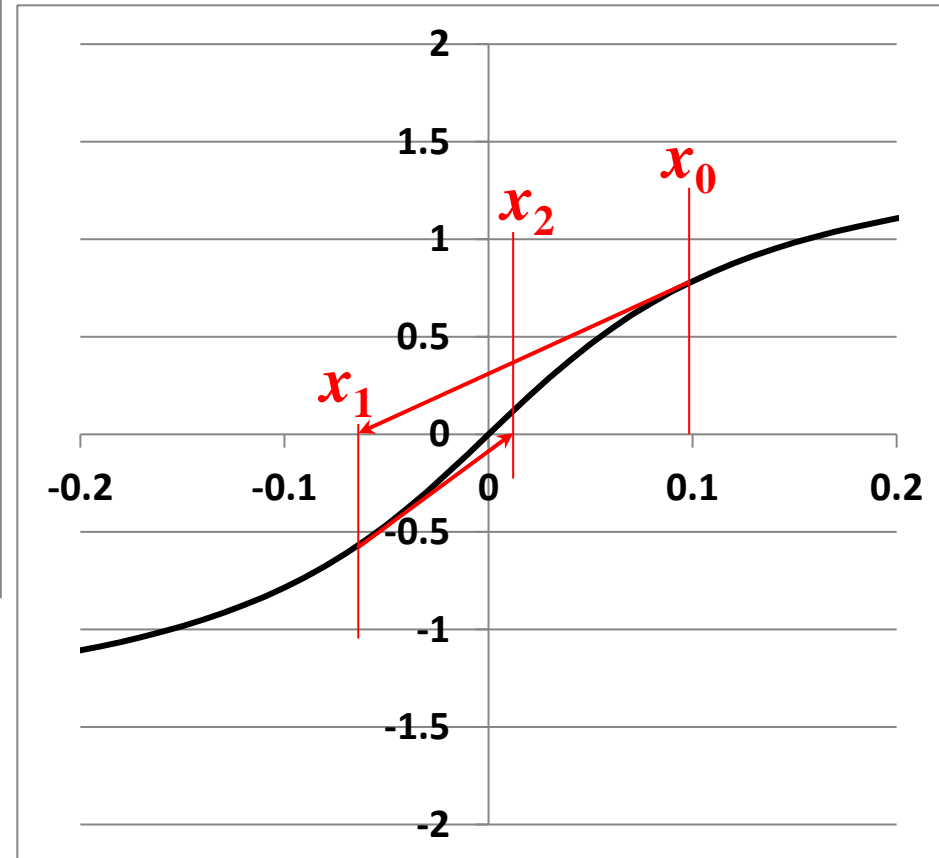
$$f(x) = \tan^{-1}(10x)$$

initial  $x = 0.1$



**A case to reach convergence**

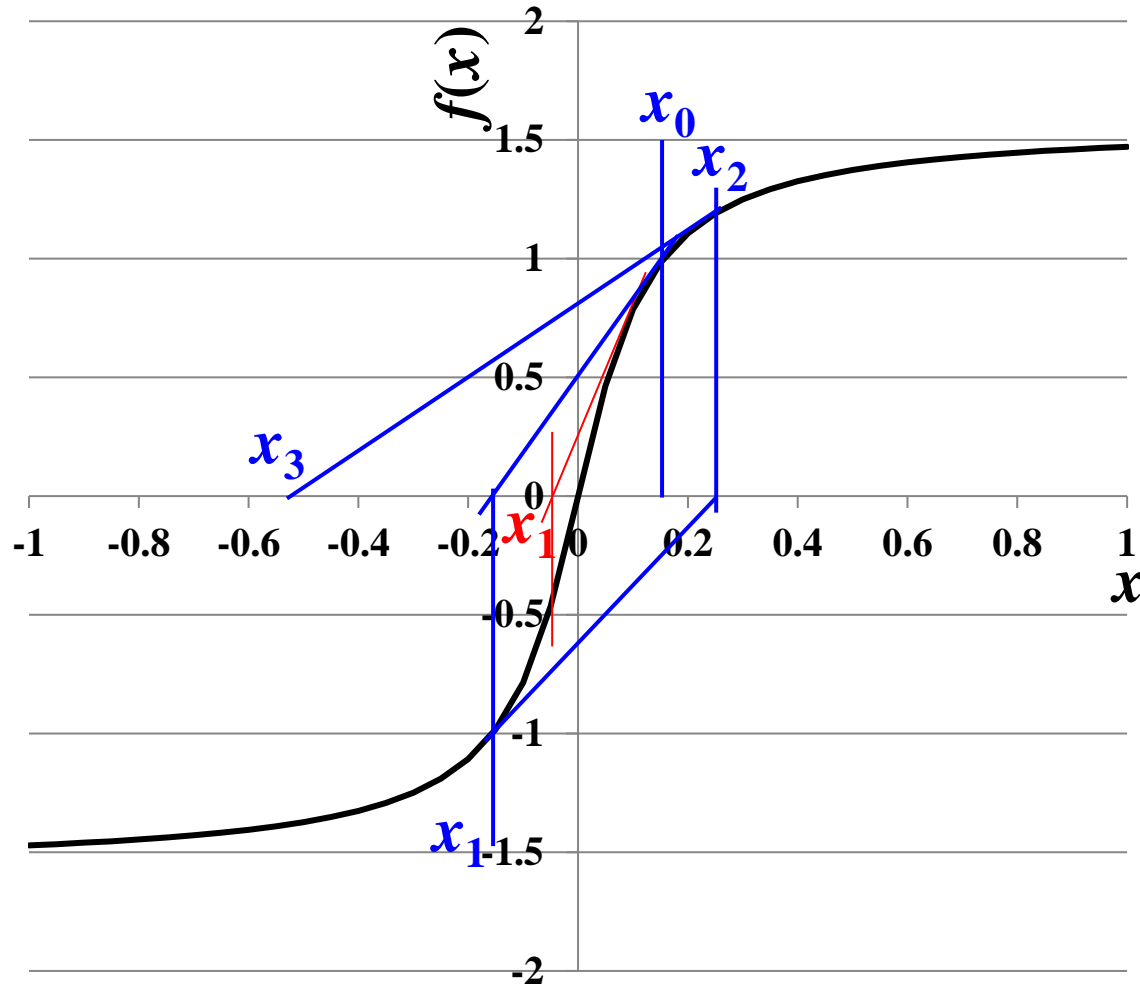
i	x	f(x)	df/dx	dx
0	0.1	0.7854	5	-0.1571
1	-0.05708	-0.5187	7.54257	0.06877
2	0.011686	0.11633	9.86527	-0.0118
3	-0.00011	-0.0011	9.99999	0.00011
4	1.15E-10	1.2E-09	10	-1E-10



# 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

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

$\alpha_k$ : **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  $\Delta E$  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  $\Delta E$  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

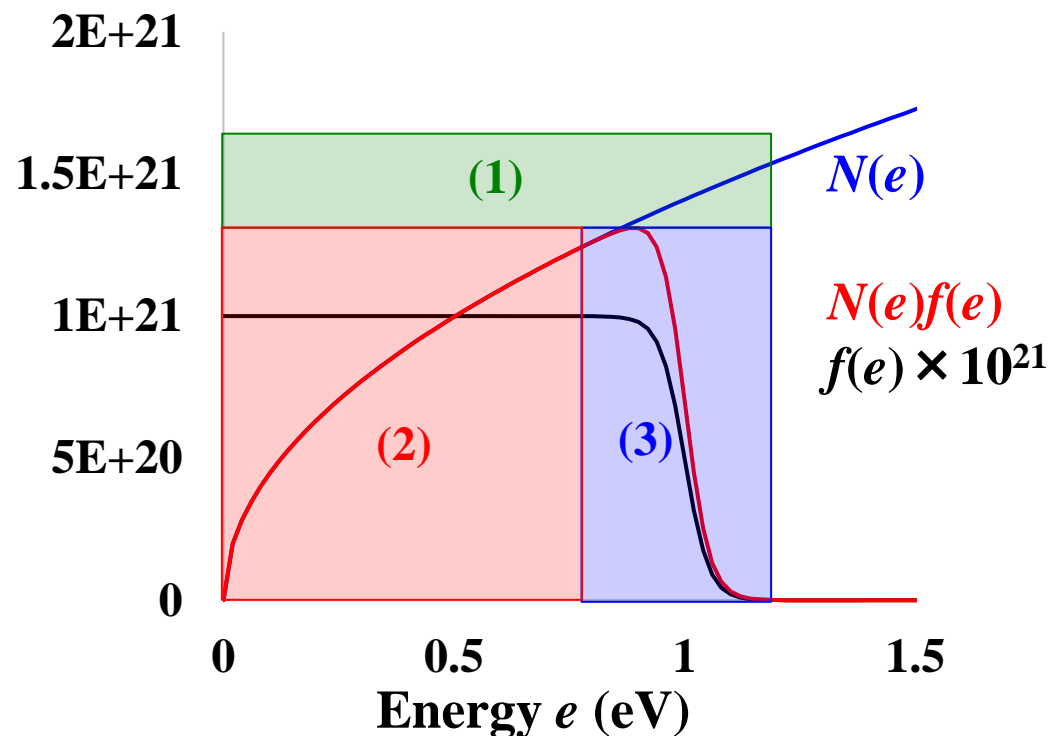
Time is measured for 300 cycles calculation

**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.

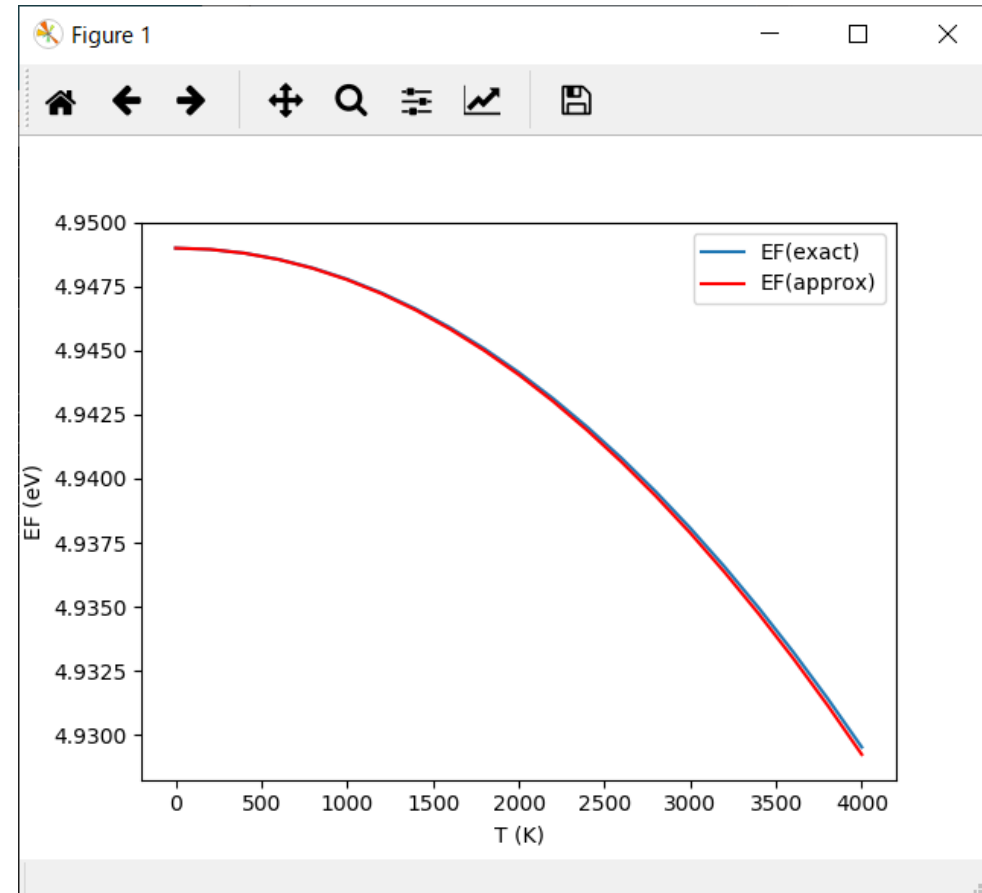
Compare with the approx. form  $E_F(T) = E_F(0) - \frac{\pi^2}{6} (k_B T)^2 N'(E_F(0))/N(E_F(0))$

Program: EF-T-metal.py

Ex.: python EF-T-metal.py

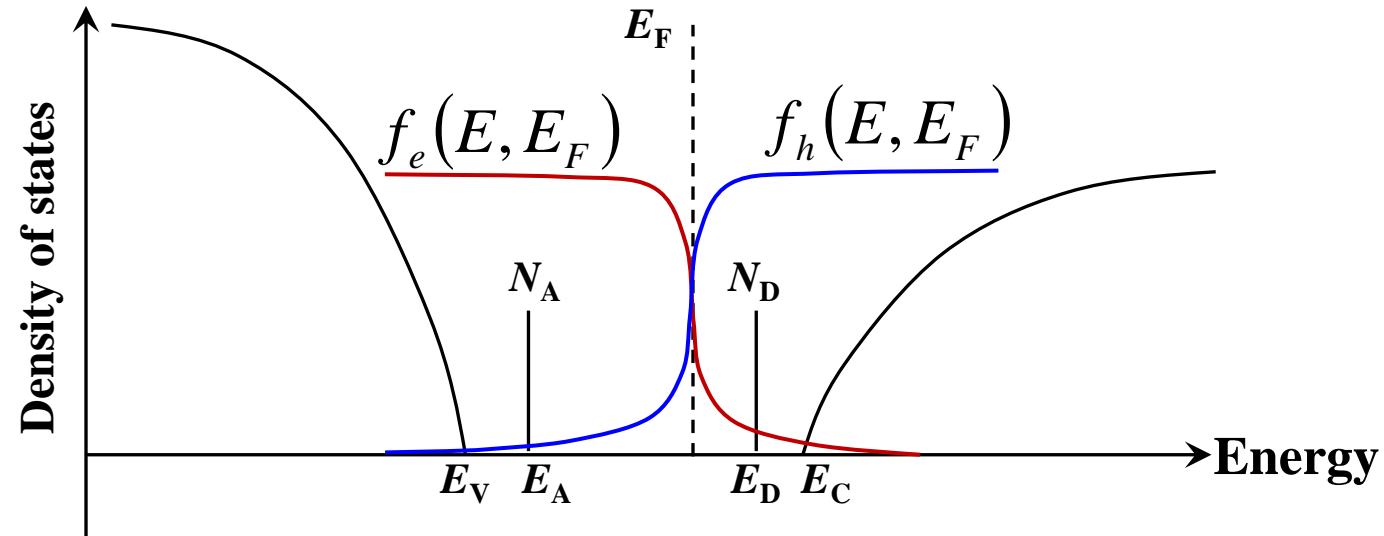
For  $n_e = 5 \times 10^{22} \text{ cm}^{-3}$

$T$ (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

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

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

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

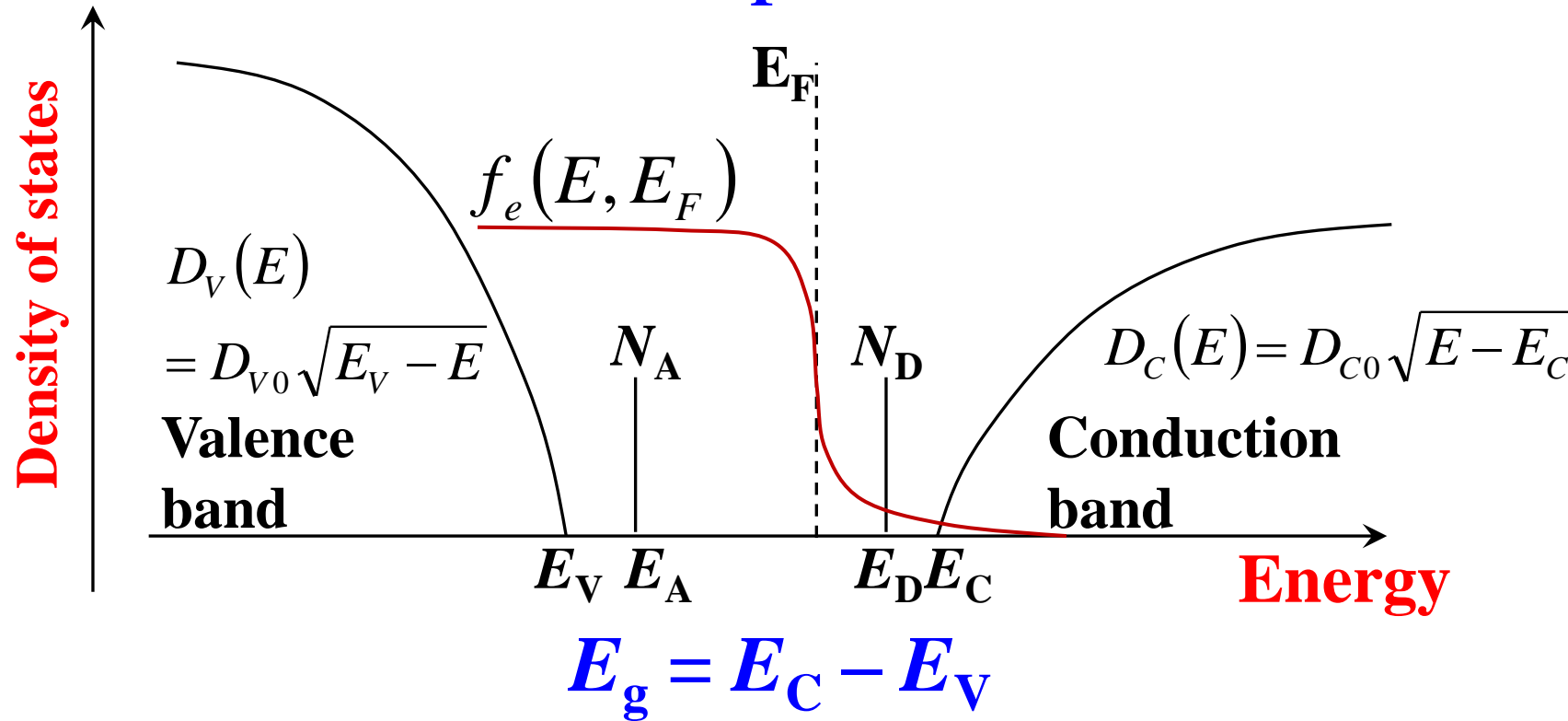
Free electron density

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

Ionized donor density

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

# How to determine $E_F$ for semiconductors



## Charge neutrality condition

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

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

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

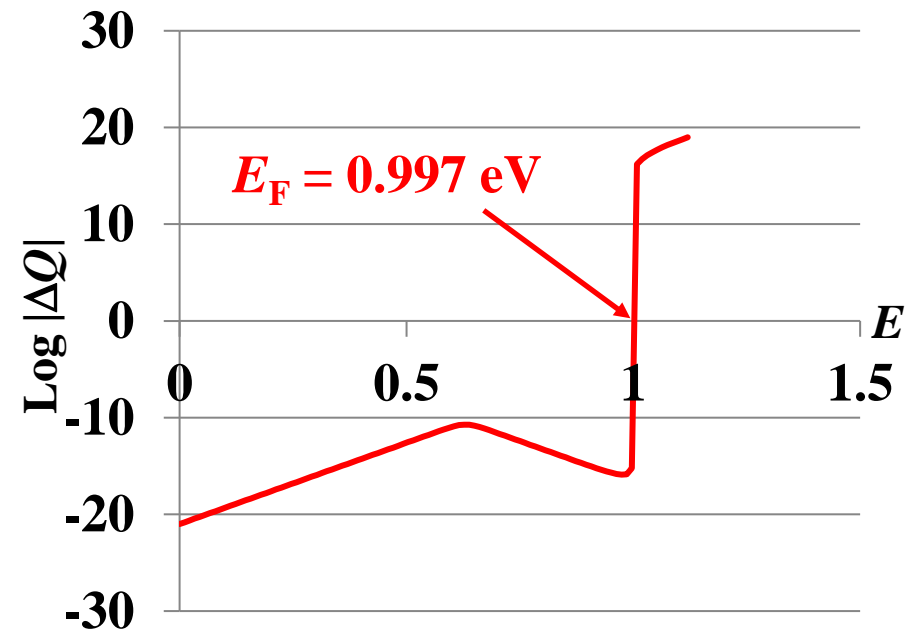
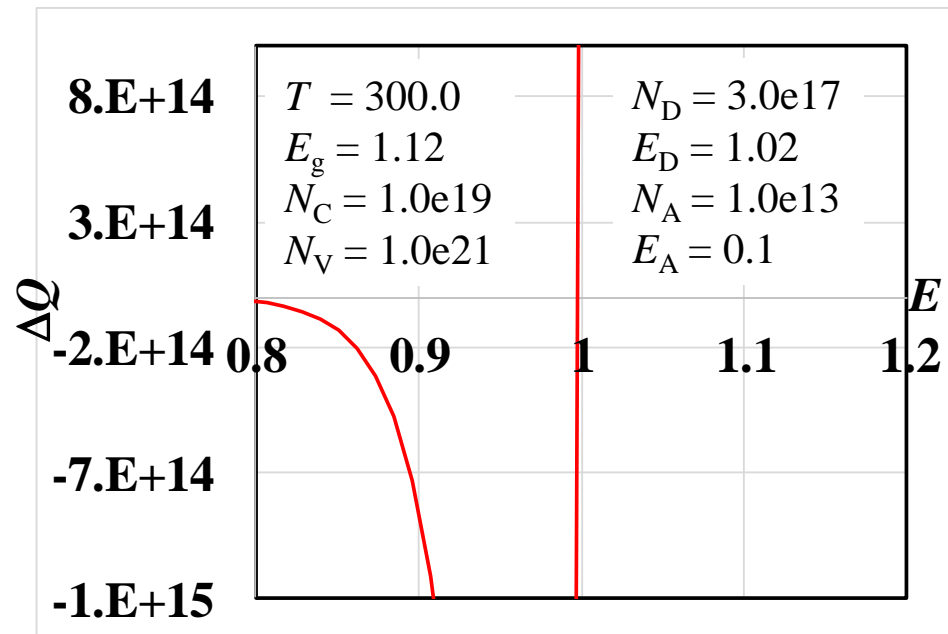
# How to calculate $E_F$ : Illustrative solution

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

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

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

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

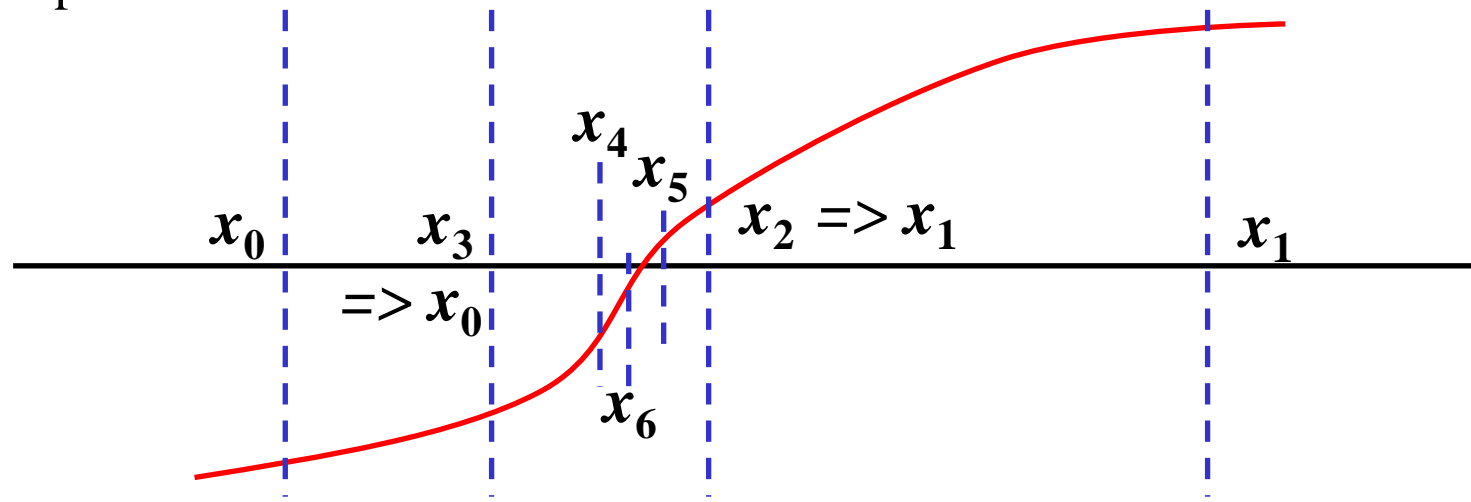
1.  $x_2 = (x_0 + x_1) / 2.0$

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  $|x_1 - x_0|, |f(x_2)|$  becomes less than EPS.

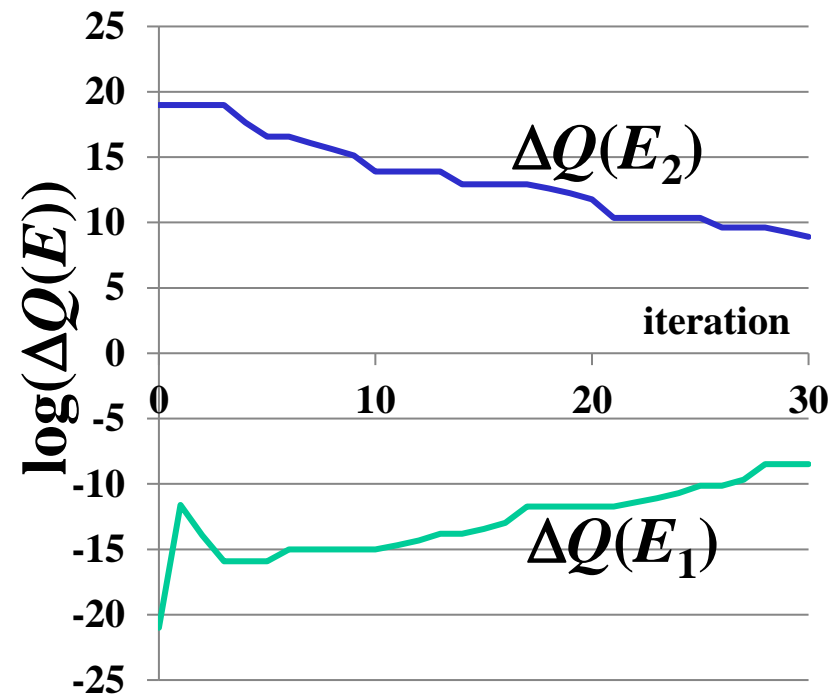
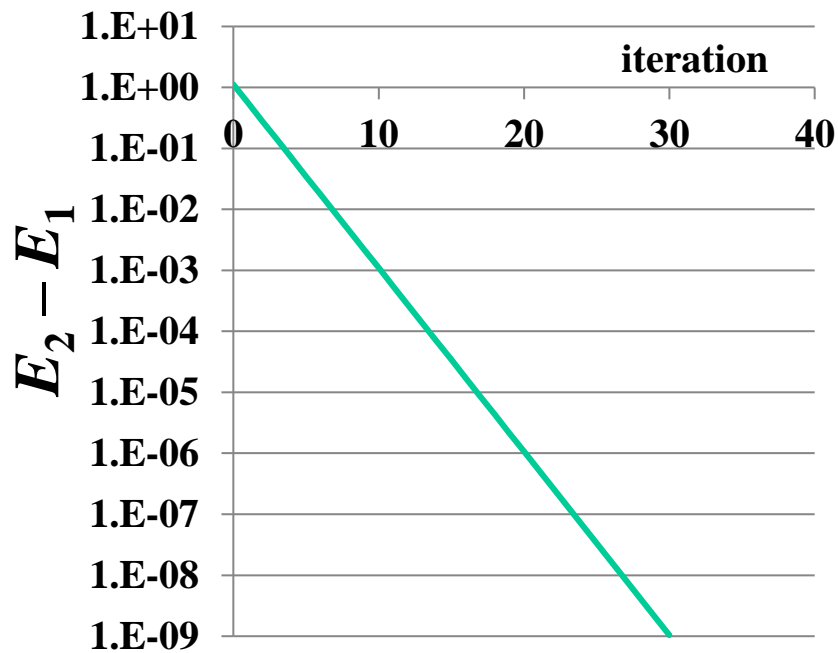
4. Repeat 1 – 3



# $E_F$ 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`

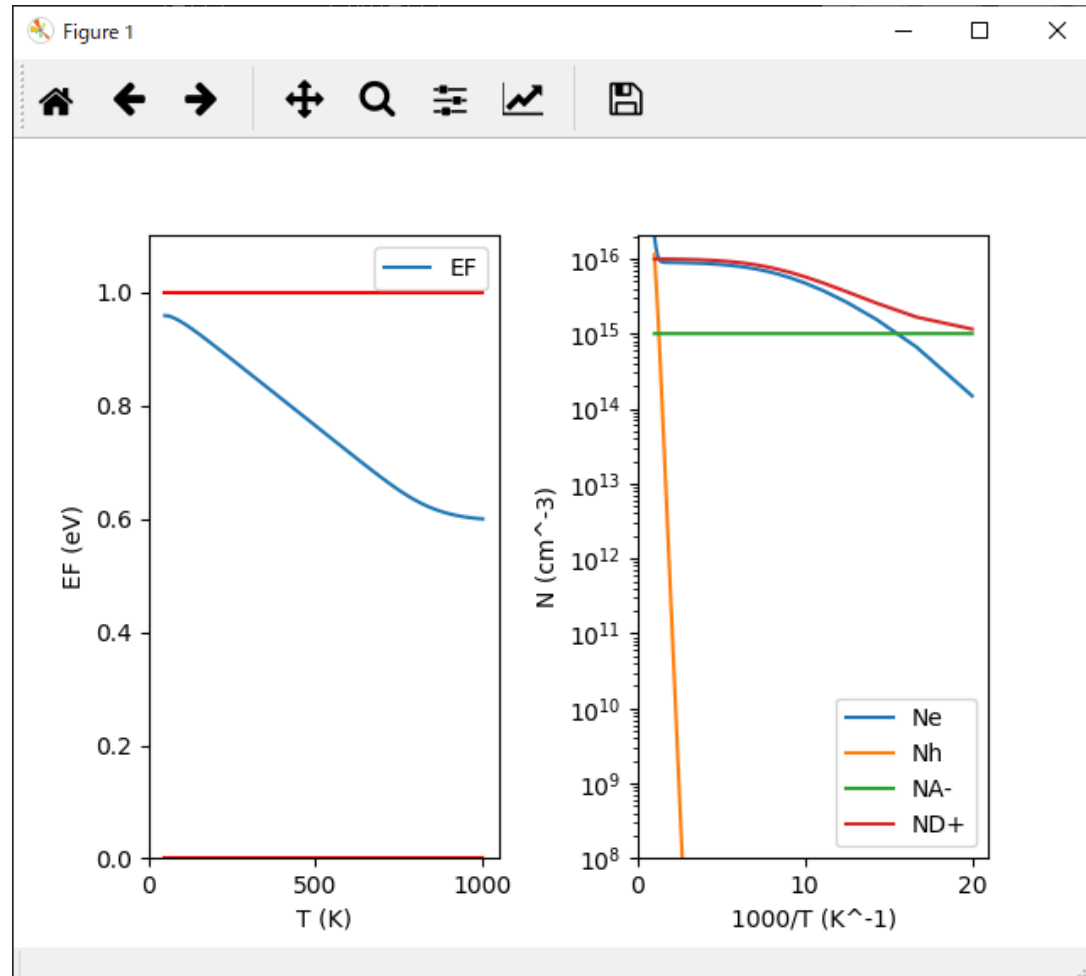
$E_V = 0$ ,  $E_c = 1.0$  eV (= band gap)

$E_A = 0.05$  eV,  $N_A = 10^{15}$  cm<sup>-3</sup>

$E_D = 0.95$  eV,  $N_D = 10^{16}$  cm<sup>-3</sup>

$N_c = 1.2 \times 10^{19}$  cm<sup>-3</sup>

$N_v = 2.1 \times 10^{18}$  cm<sup>-3</sup>



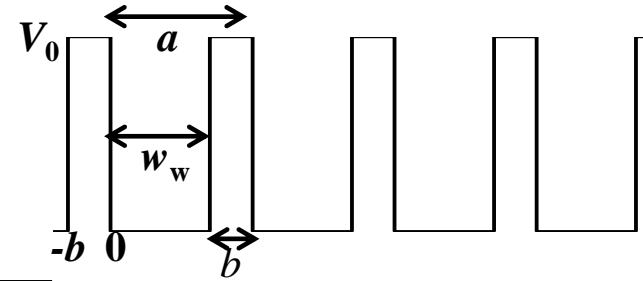
# Multi-valued equation: Kronig-Penney model

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:

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

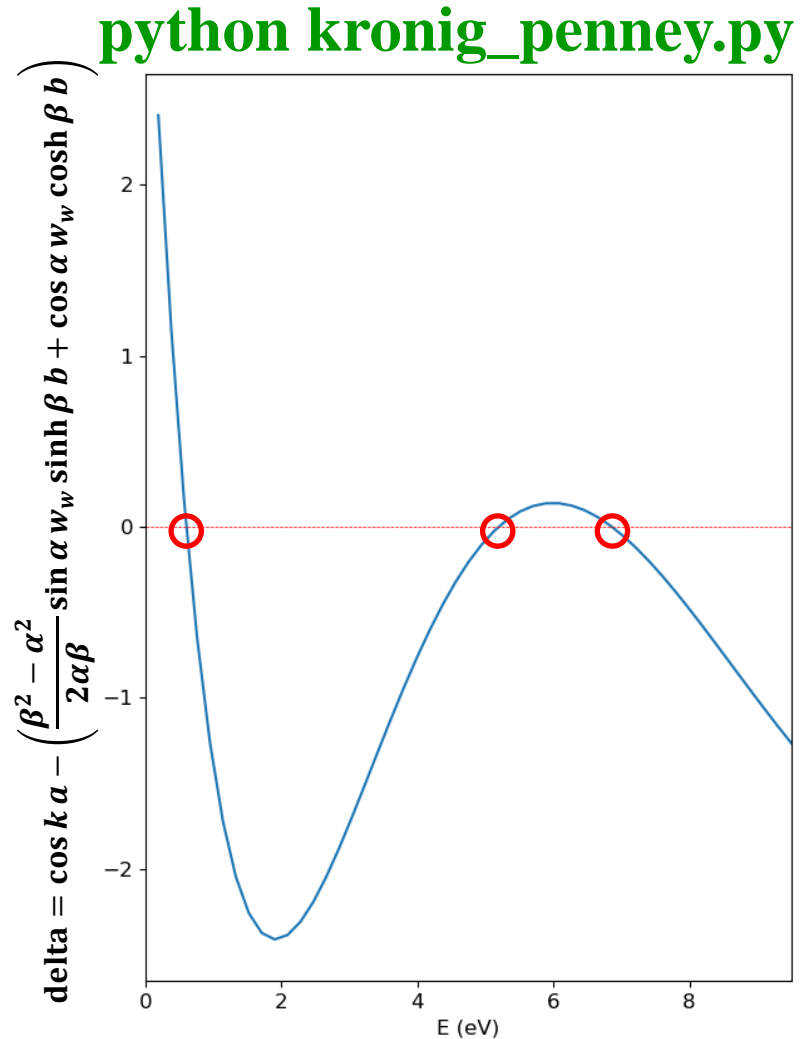
**Scan  $E$  in possible range to find all the solutions,  
then use them for initial values  
to obtain accurate values by Newton-Raphson method**

# Program: Kronig-Penney model

Program: `kronig_penney.py`

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

Barrier width  $0.5 \text{ \AA}$       Barrier height  $10.0 \text{ eV}$



python `kronig_penney.py band`

