

Nonlinear (NL) optimization

非線形最適化

Linear problem vs Nonlinear problem

Linear least-squares:

- Applied to model functions whose unknown parameters enter linearly
ex: $f(x) = a_0 + a_1 * \exp(-x) + a_2 / x$
- **No initial guess is required (The solution is obtained by a single matrix calculation)**

Nonlinear optimization / least-squares:

- Applied to model functions whose **unknown parameters enter nonlinearly**
ex: $f(x) = a_0 + a_1 * \exp(-a_2 * x)$
ex: $f(x) = a_0 + a_1 / (x + a_2)$
- **Initial guess of parameters is required**
- The problem is solved iteratively, e.g., by local linear approximation using Taylor expansion
- The number of iterations required for convergence is difficult to estimate
- **A poor initial guess may lead to a local minimum, divergence, or oscillation**

Good initial parameters are important

Optimization

Objective: Find parameters x_i to minimize or maximize an objective function $F(x_i)$

Maximization problem for $F(x_i)$

is equivalent to minimization problem for $-F(x_i)$

We will focus on minimization problems

Examples:

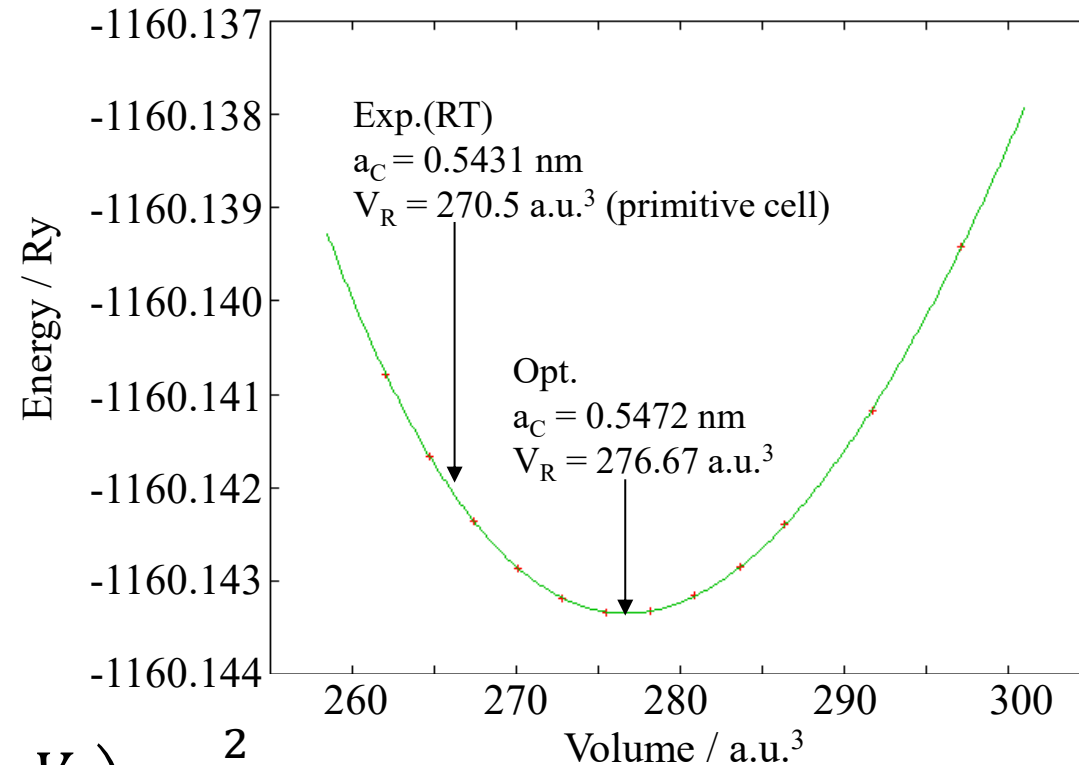
- **Linear least-squares method:** A linear minimization problem for L2 norm of errors
- **Curve fitting:** A nonlinear minimization problem for L2 norm of errors
- **Structure relaxation:** A nonlinear minimization for total energy

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



$$E = B_0 \left(\frac{V - V_0}{V_0} \right)_{\min}^2$$

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

Profile models used for spectroscopy

Lorentz function

$$I_L(x) = \frac{1}{1 + [(x - x_0)/w]^2} \quad \mathbf{w: \text{half width at half maximum}}$$

Gauss function

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

$a_w = (\ln 2)^{-1/2} = 0.832554611$

Voigt function:

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

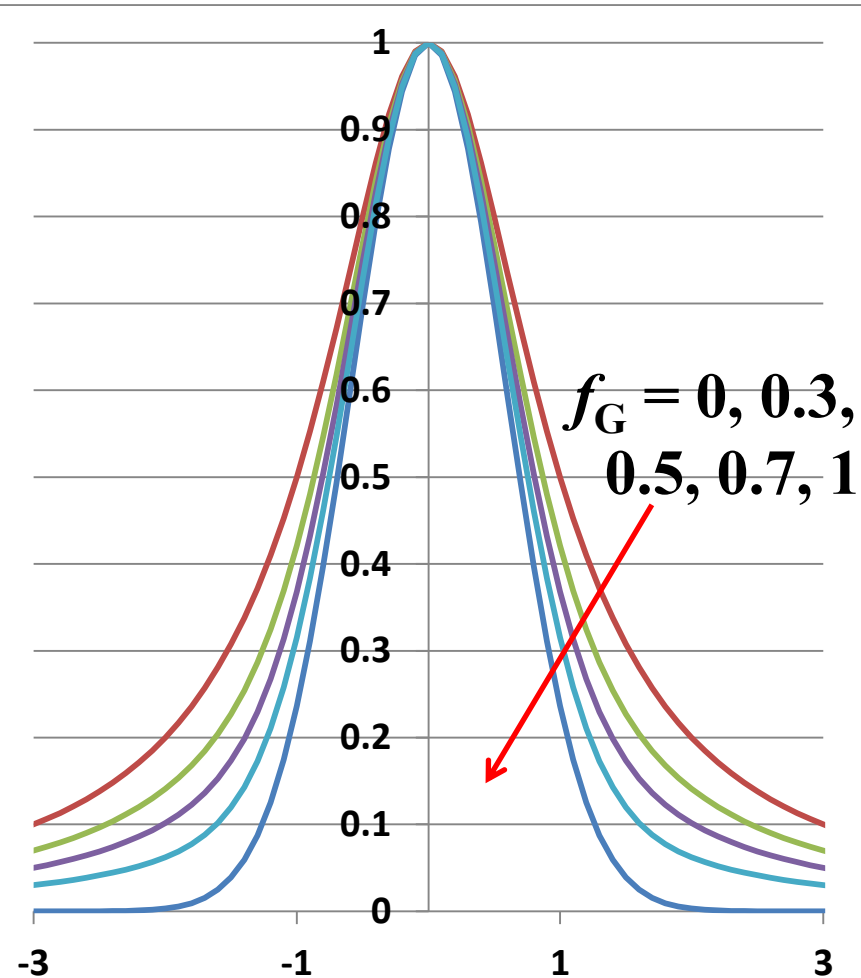
$$I_V(x) = \int_{-\infty}^{\infty} I_G(x') I_L(x - x') dx'$$
$$= \frac{a_V}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x'^2)}{a_V^2 + (x - x')^2} dx'$$

Pseudo-Voigt function:

Simplified Voigt function

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

$\mathbf{f_G: \text{Gauss fraction}}$



Methods of nonlinear (NL) optimization

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

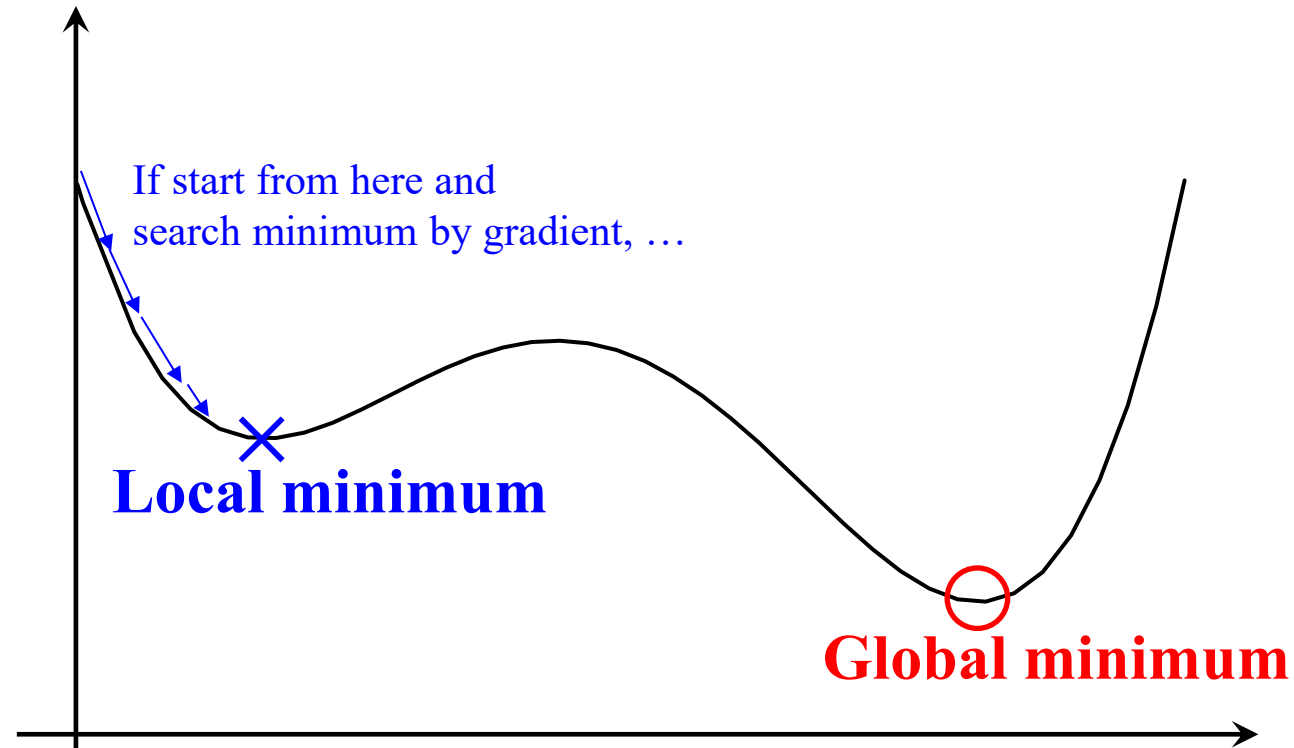
Direct search method (直接探索法)

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:

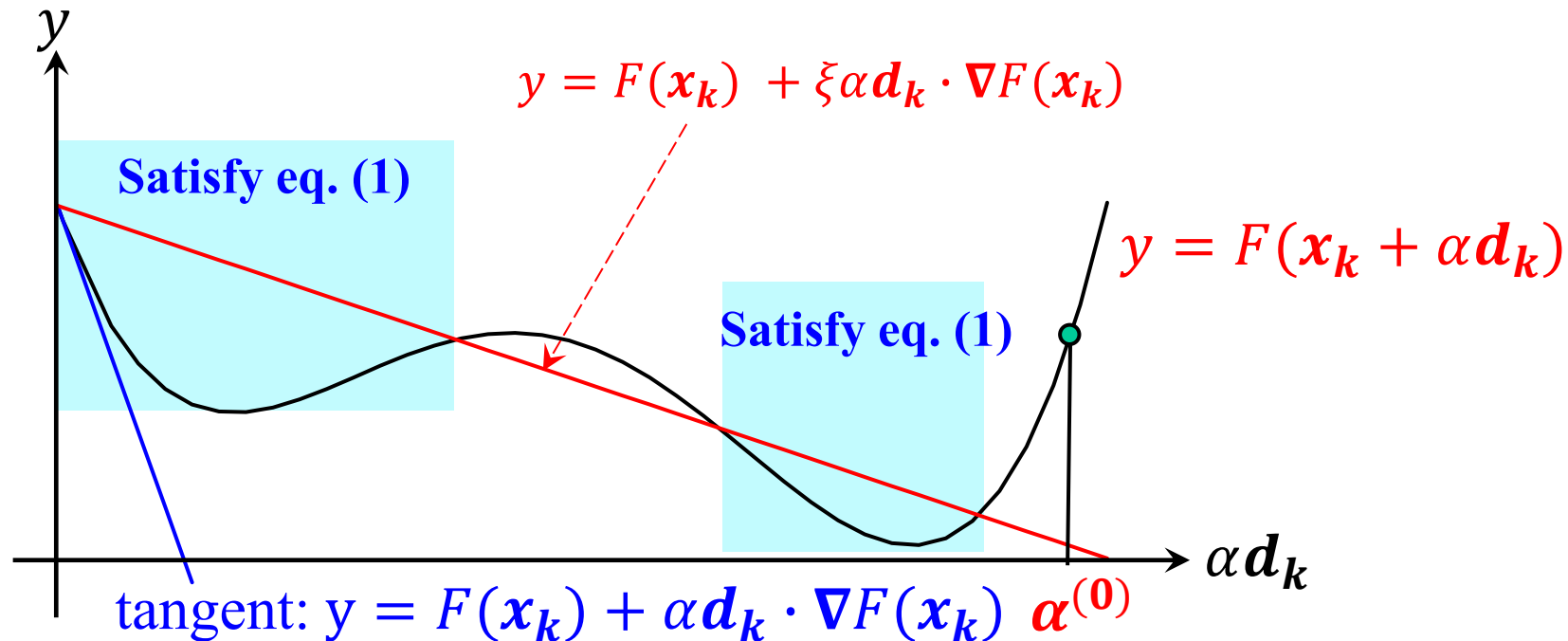
- 1. Employ a large initial search range**
- 2. Not use a direct value of gradient**

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 algorithm:

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

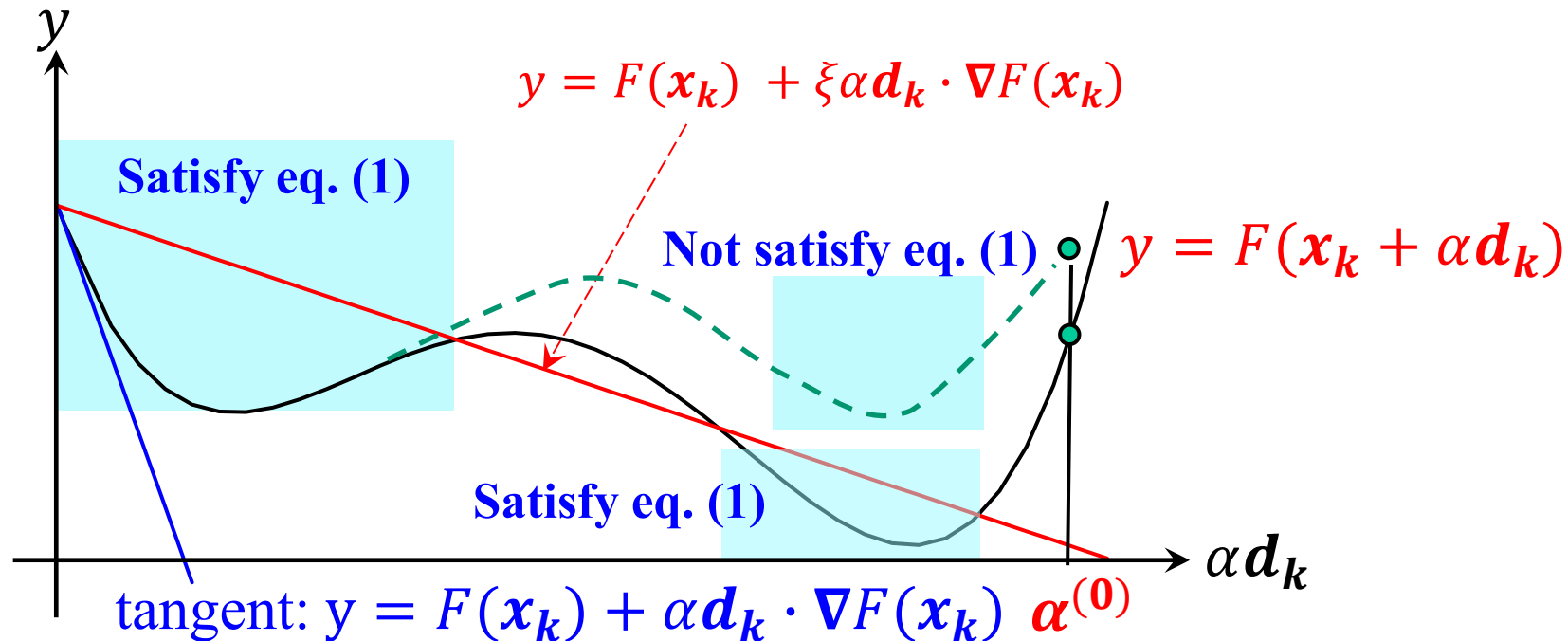


Line search (直線探索法): Armijo condition

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 algorithm:

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

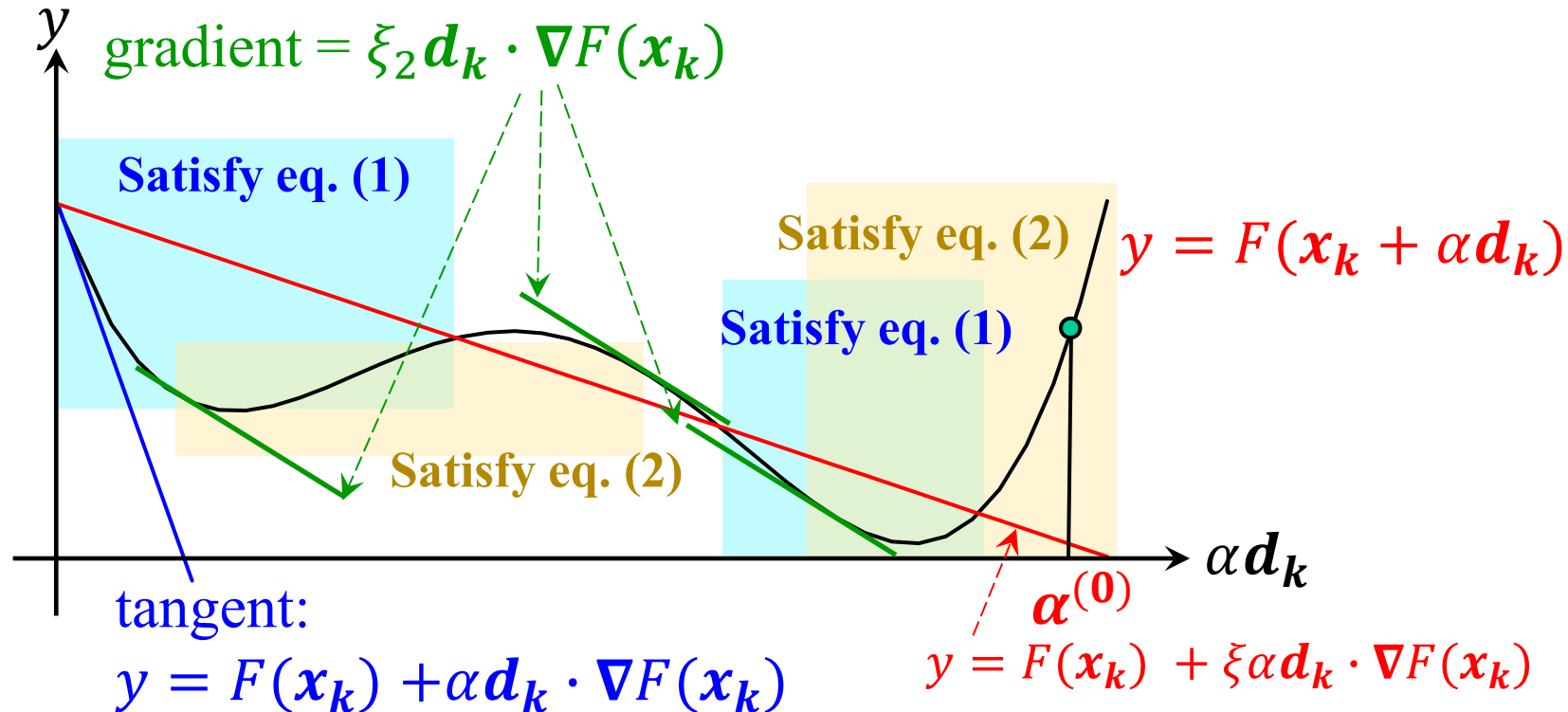
矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

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 monotonic 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 $f(x)$

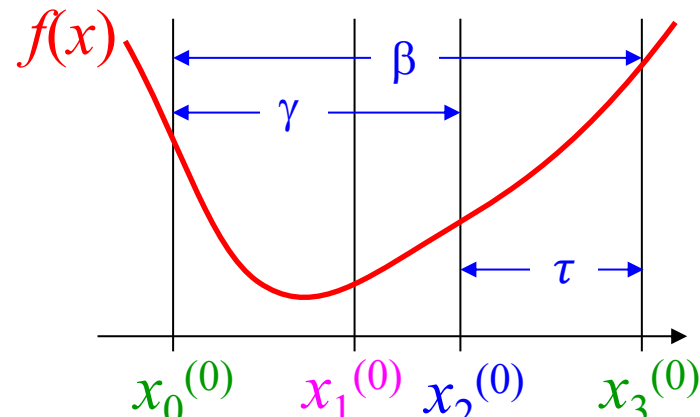
Unique solution exists in the range $[x_0^{(0)}, x_3^{(0)}]$ if $f(x_1^{(0)}) < f(x_0^{(0)}), f(x_3^{(0)})$ for $x_0^{(0)} < x_1^{(0)} < x_3^{(0)}$

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

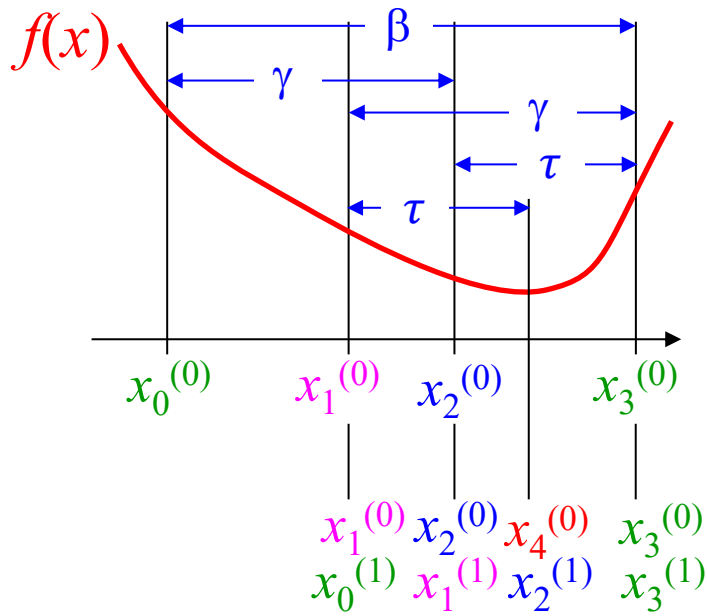
Replace $x_2^{(0)}$ with $x_4^{(0)}$ in $[x_0^{(0)}, x_1^{(0)}]$

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)

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

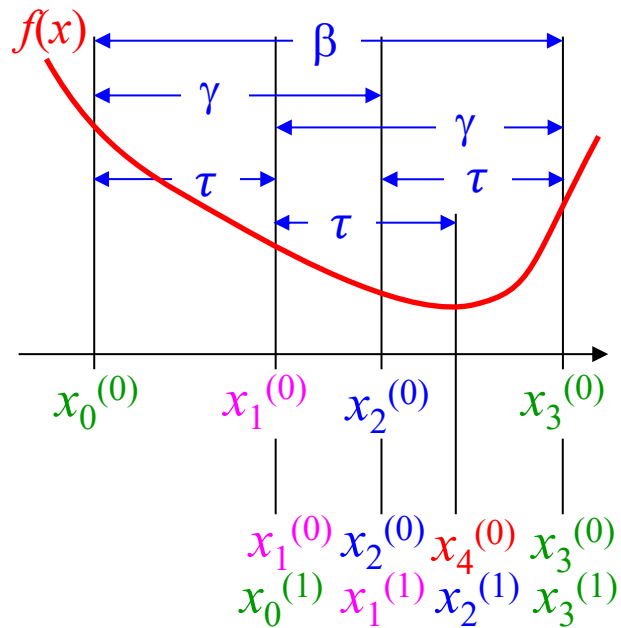
$$= x_3^{(k)} - x_1^{(k)} : x_4^{(k)} - x_1^{(k)} = \gamma : \tau$$

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

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

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 nonlinear (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:

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

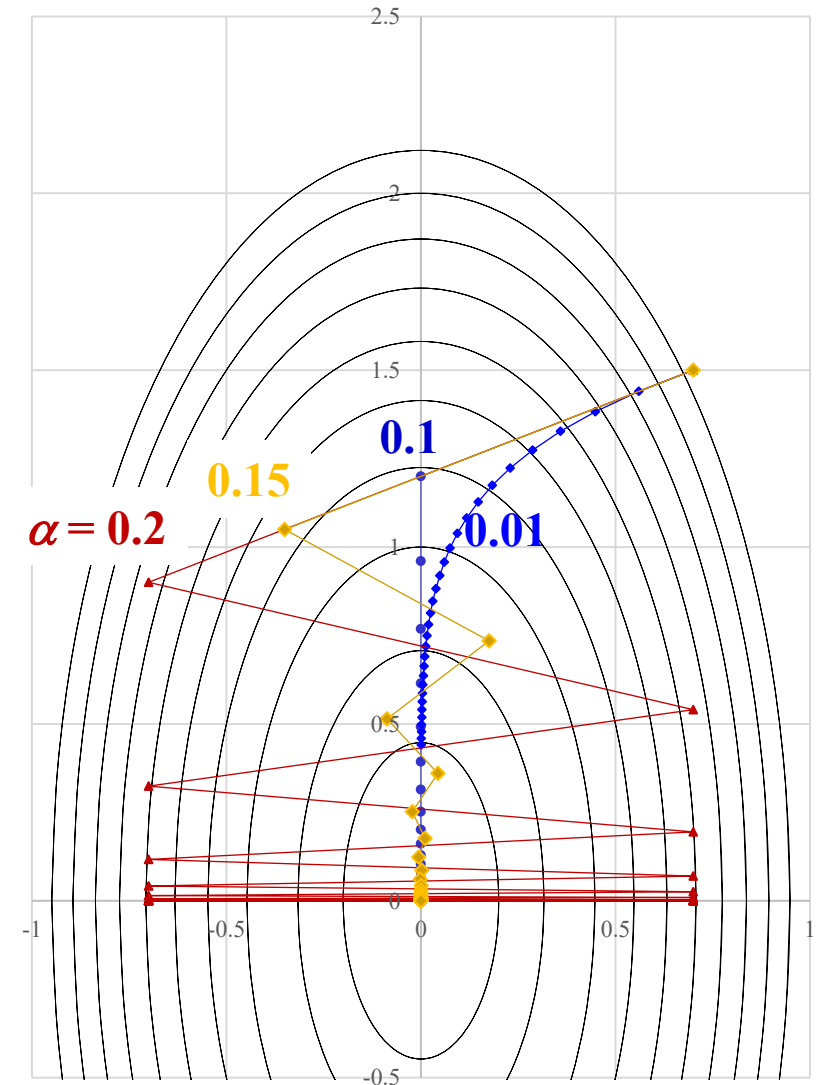
▪ **SD method**

- $\alpha = 0.3$: Diverged (not shown in the graph)
- 0.2 : Oscillated, not converged
- 0.15 : Converged, but oscillated
- 0.1 : Reach final x_1 by one cycle calculation
- 0.01 : Not oscillated, but slowly converged

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

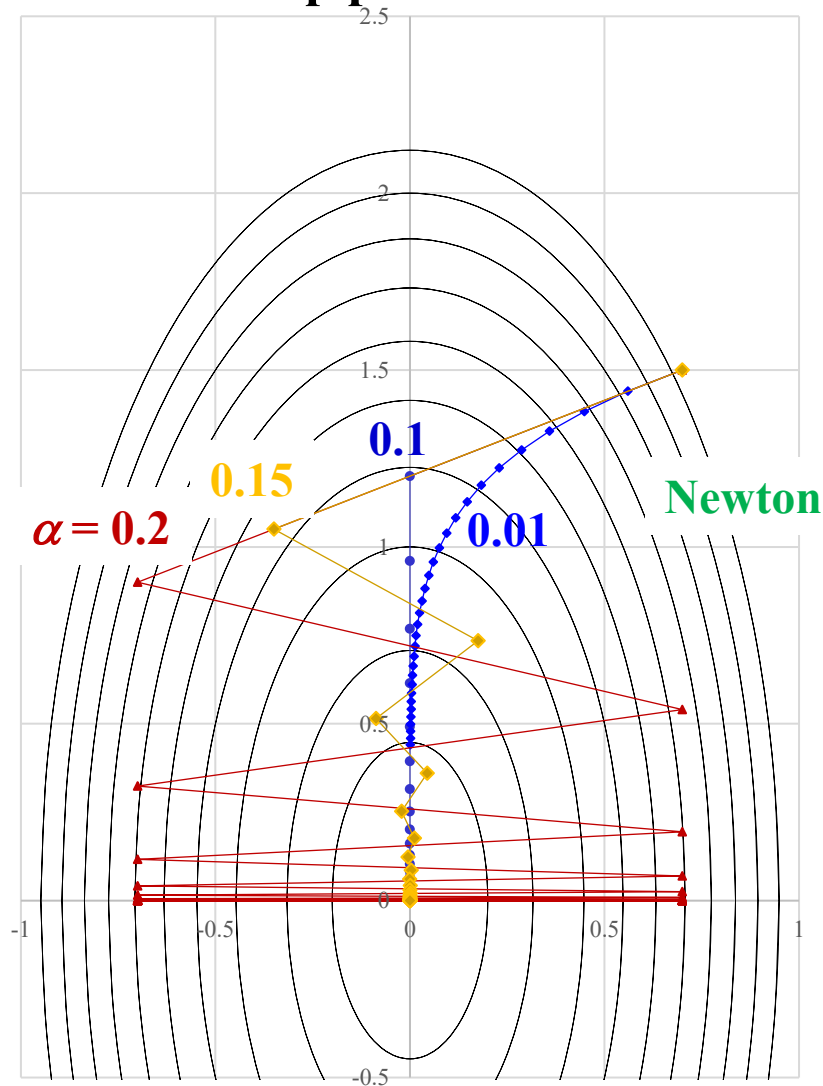
が大きく非対称な場合、最急勾配方向は最小値方向とは大きく異なることがある

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

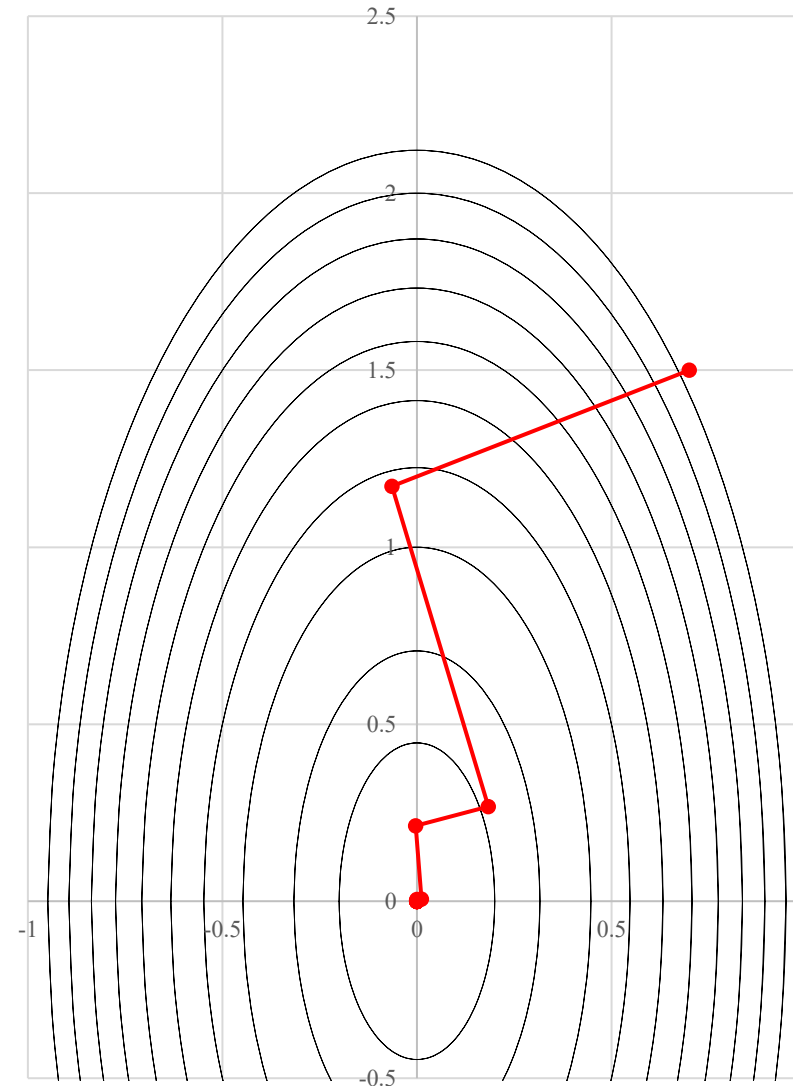


Steepest descent method

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:

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

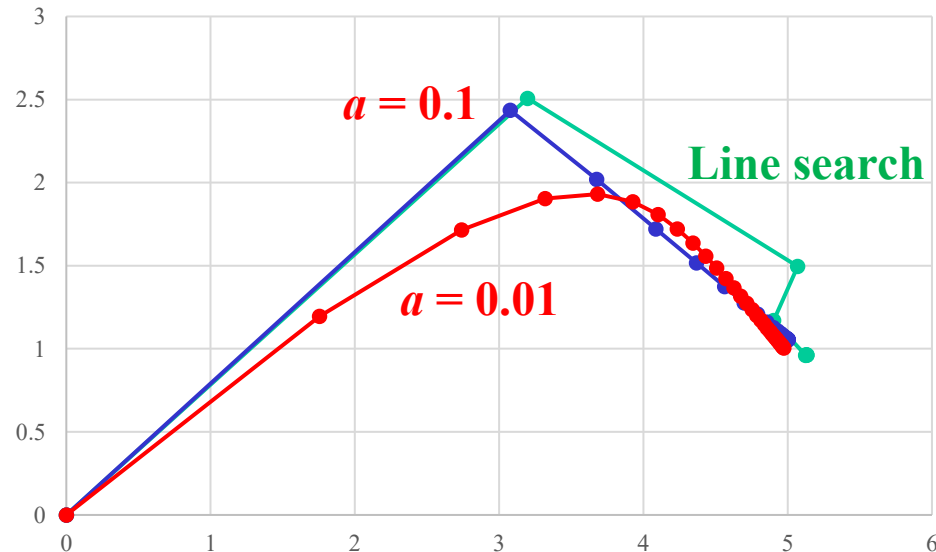
1000 batch data $(x_{1i}, x_{2i}, f(x_i))$ are generated at random

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

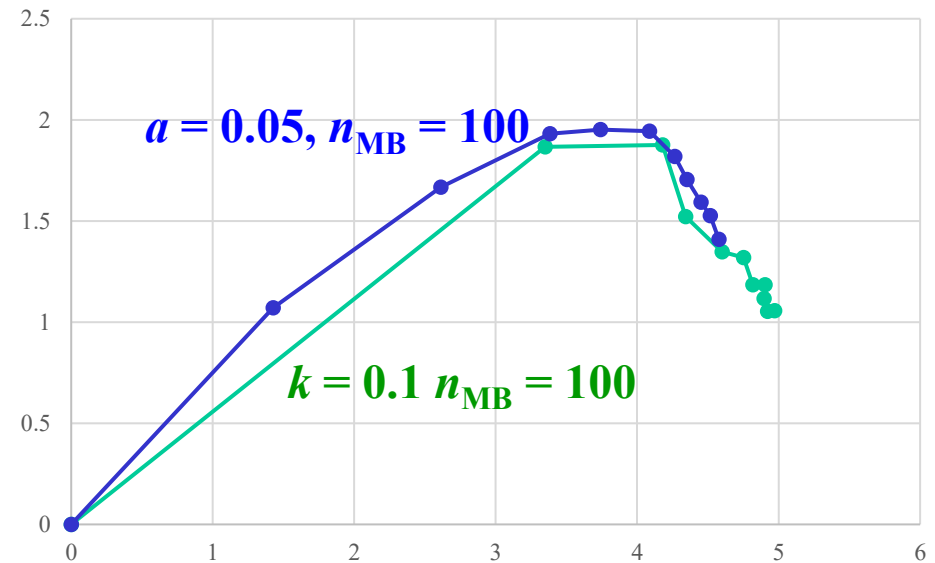
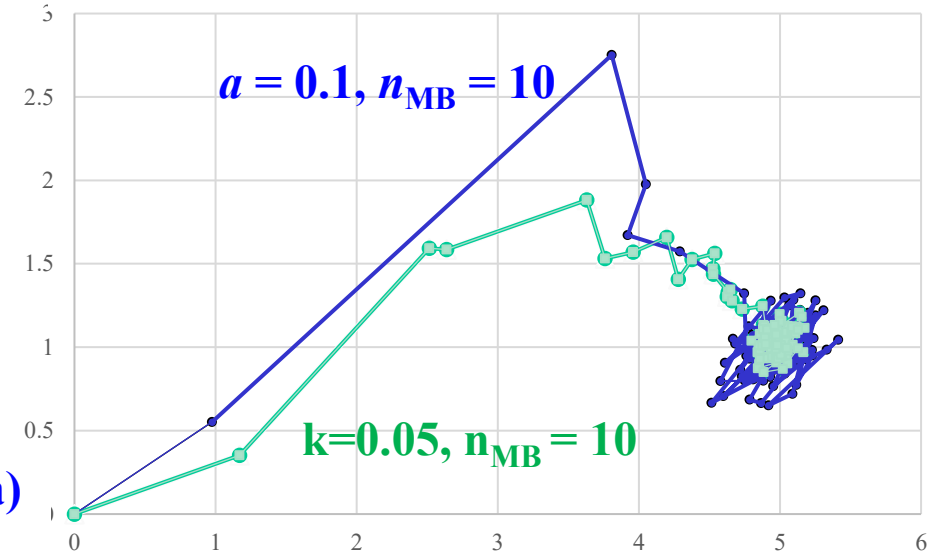
Estimate a and b

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

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

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

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

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 : Damping Factor

SD vs. Newton-Raphson methods

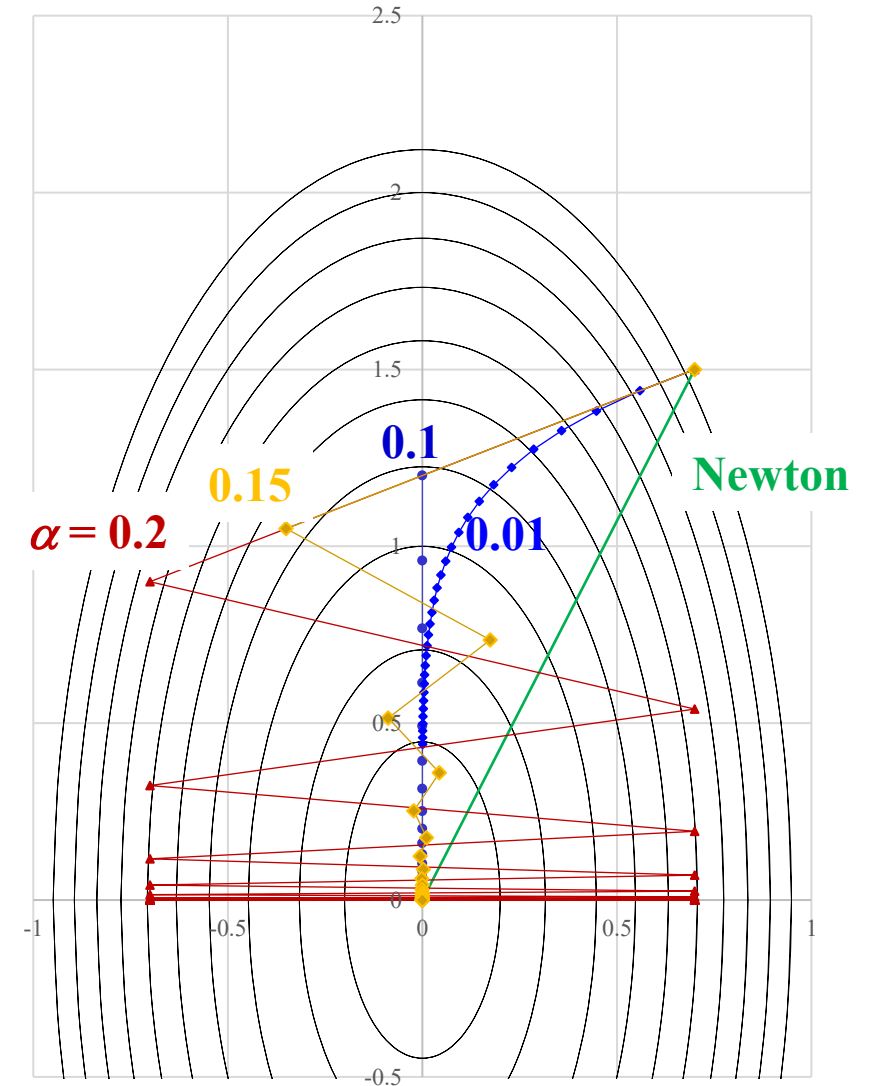
- **SD method**

Steepest direction \neq Optimization direction

- **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

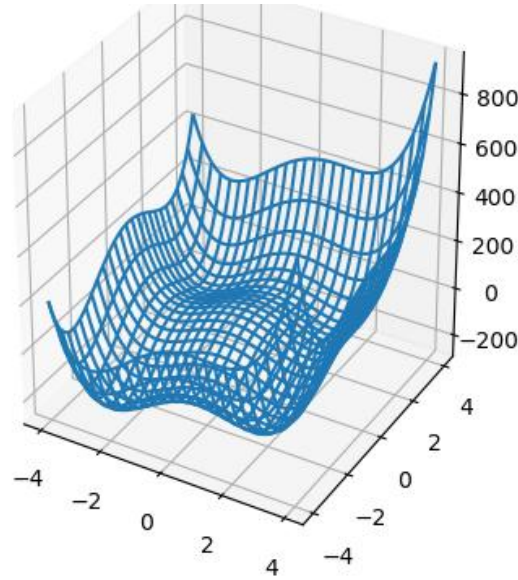


Improve SD method to follow optimization directions

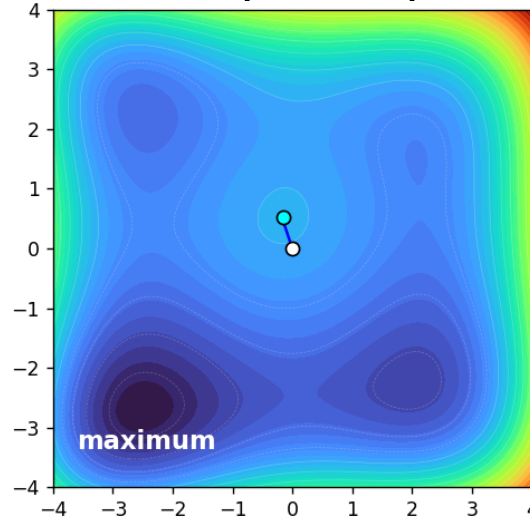
Problem of Newton method: optimize-newton-raphson2d.py

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

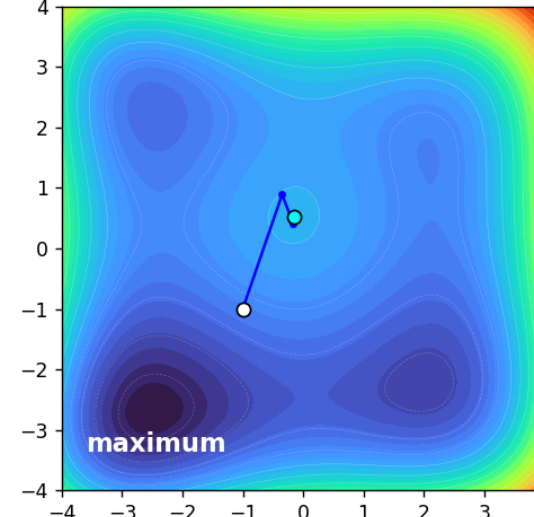
Usage: `python optimize2d.py --mode newton --x0=x0 --y0=y0`



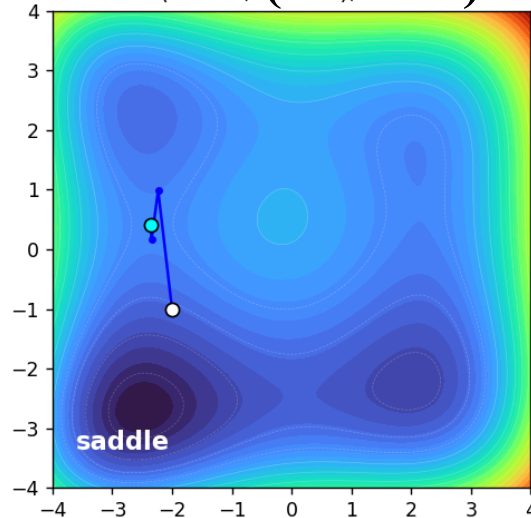
From (0.0 0.0)



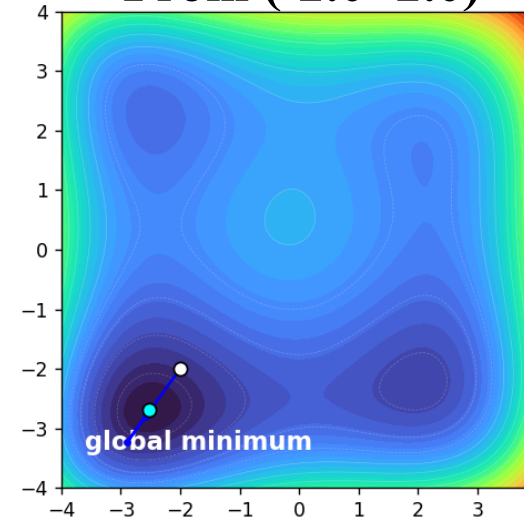
From (-1.0 -1.0)



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)

Objective 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 the Hessian matrix is costly because it contains second derivatives
- (2) Eigenvalues of the Hessian matrix can be negative
=> may lead to a maximum or saddle point
- (3) Easy to diverge

Quasi-Newton method:

- (1,2) Approximate the inverse Hessian (H_i) from changes in gradients.

$$s_i = x^{(i+1)} - x^{(i)}, \quad y_i = \nabla F(x^{(i+1)}) - \nabla F(x^{(i)})$$

Inverse secant condition: $H_{i+1}y_i = s_i$

- (3) Line search algorithm is applied along the search direction: $d_i = -H_i \nabla F(x_i)$

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)

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

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

DFP method: The first formulation of quasi-Newton method

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

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

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

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

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

Algorithm:

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

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

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

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

STEP 4: End if self-consistency is achieved; or, go to STEP 5

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

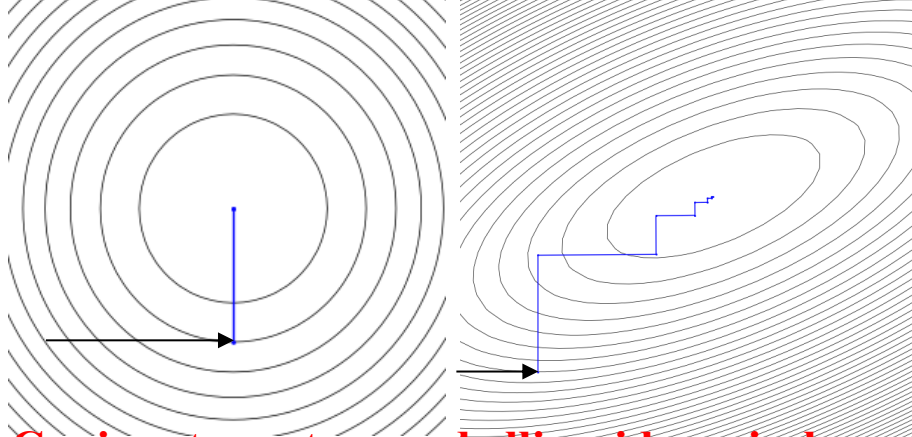
Vectors u and v satisfy $u^T A v = 0$ for a matrix A : u and v are conjugate with each other

- 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 => 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 ellipso – 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 $n_{\text{param}} = 2$).

Simple example: peakfit.py

```
from scipy.optimize import minimize
```

```
def ycal(xin, xk):
```

```
-- Cut --
```

```
    return ret
```

```
def ycal_list(xin, xk):
```

```
-- Cut --
```

```
    return y
```

```
def minimize_func(xk, xin, yin):
```

```
    nx = len(xin)
```

```
    S2 = 0.0
```

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

```
        yc = ycal(xin[i], xk)
```

```
        d = yin[i] - yc
```

```
        S2 += d * d
```

```
    return S2
```

```
def fit():
```

```
-- Cut --
```

```
    res = minimize(lambda xk: minimize_func(xk, xin, yin), x0,
```

```
                    jac = lambda xk: diff1(xk, xin, yin),
```

```
                    method = method, tol = tol,
```

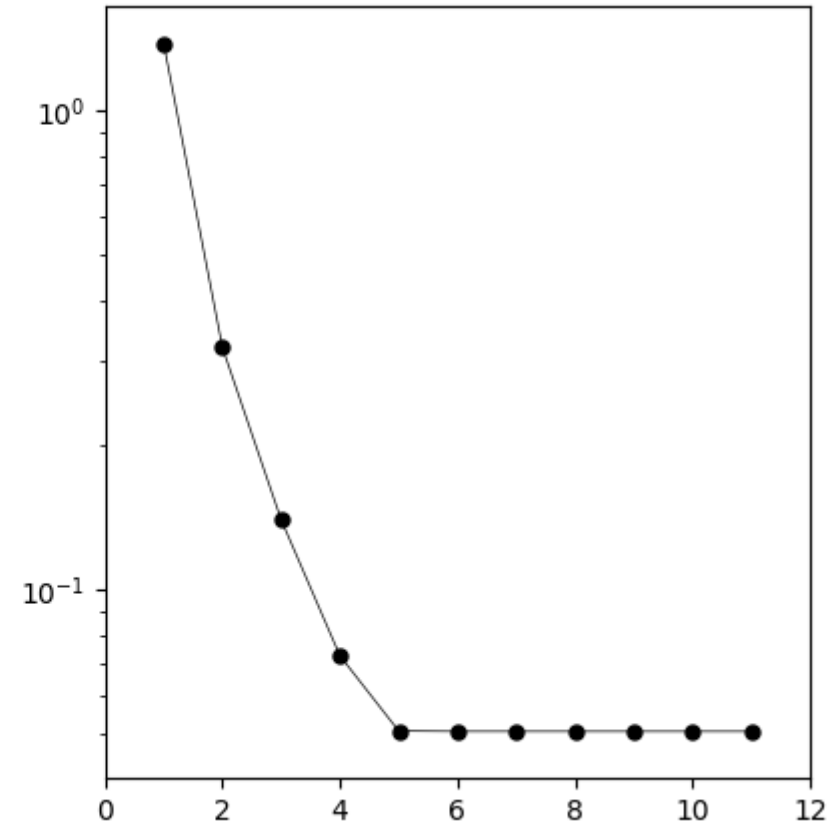
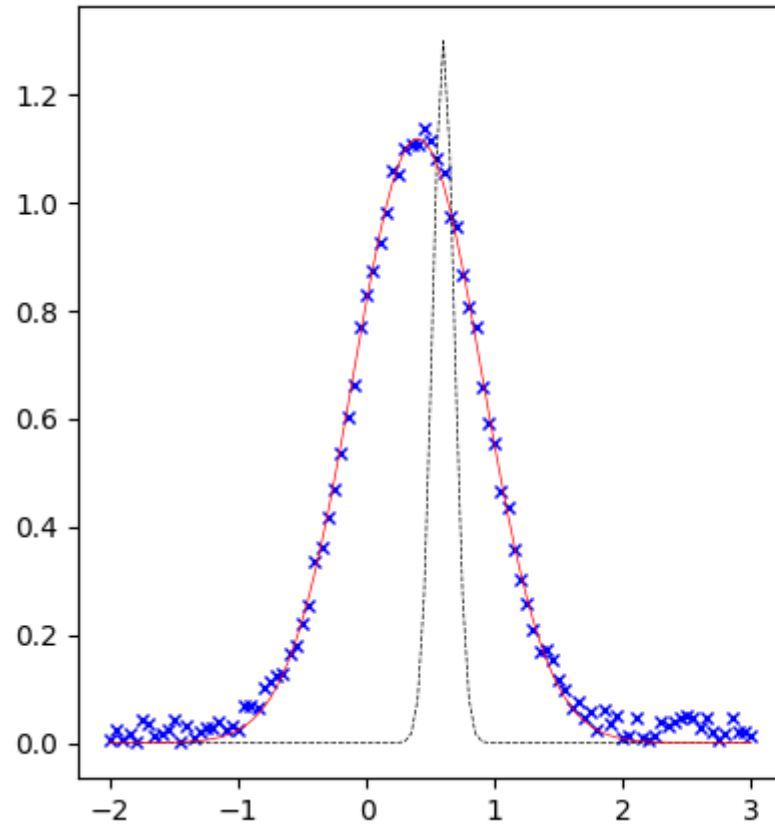
```
                    callback = lambda xk: callback(xk, xin, yin),
```

```
                    options = {'maxiter':maxiter, "disp":True})
```

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`)

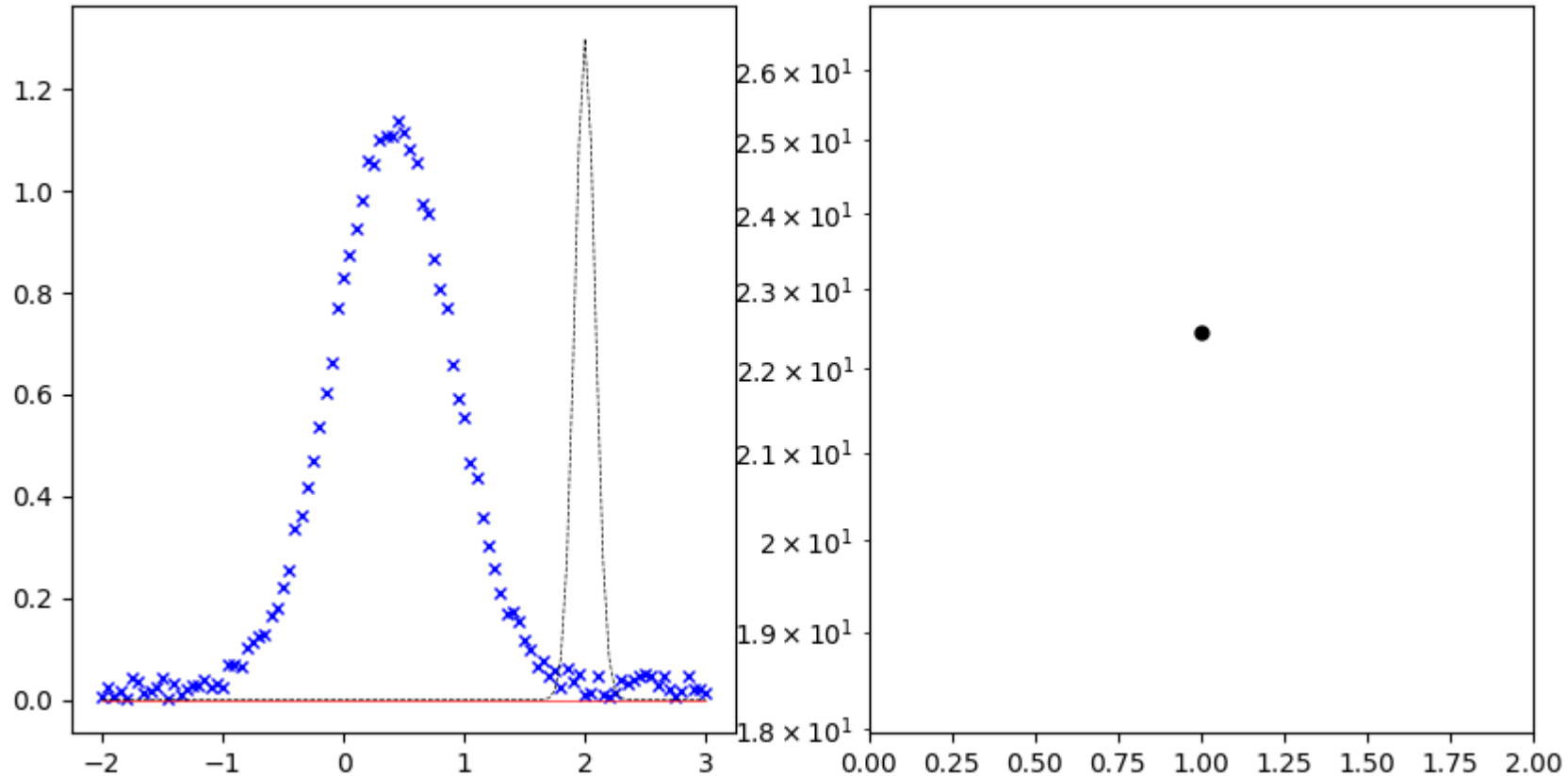


Not optimized

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

method: cg

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

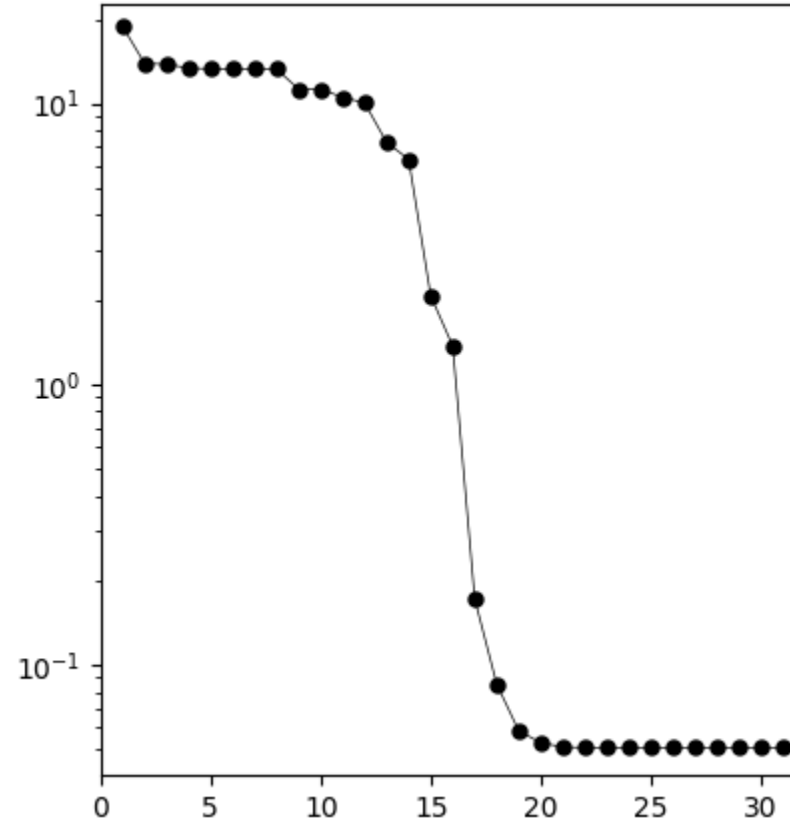
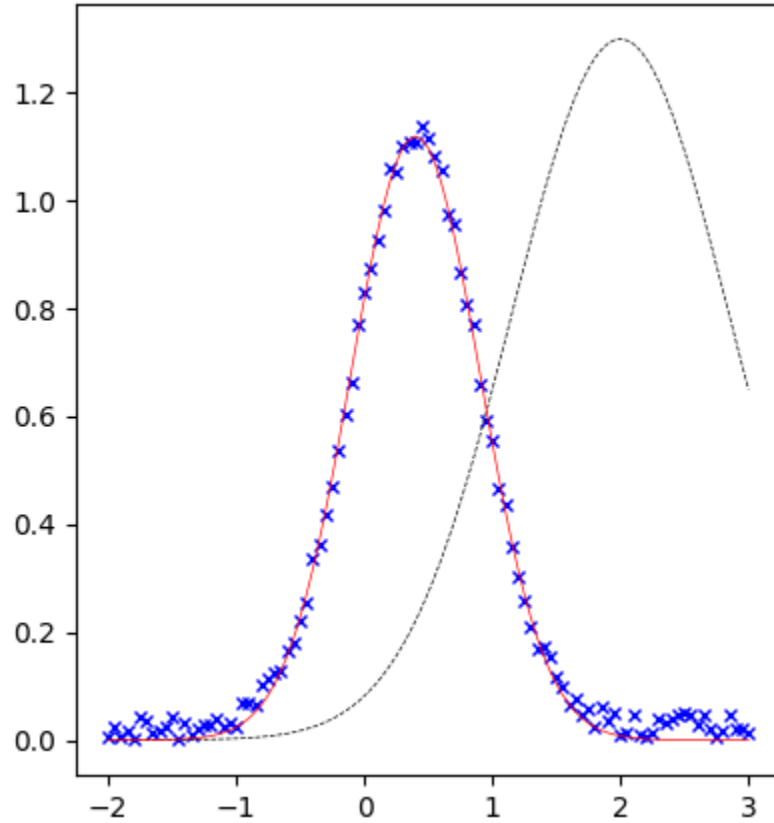


Converged

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

method: cg

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



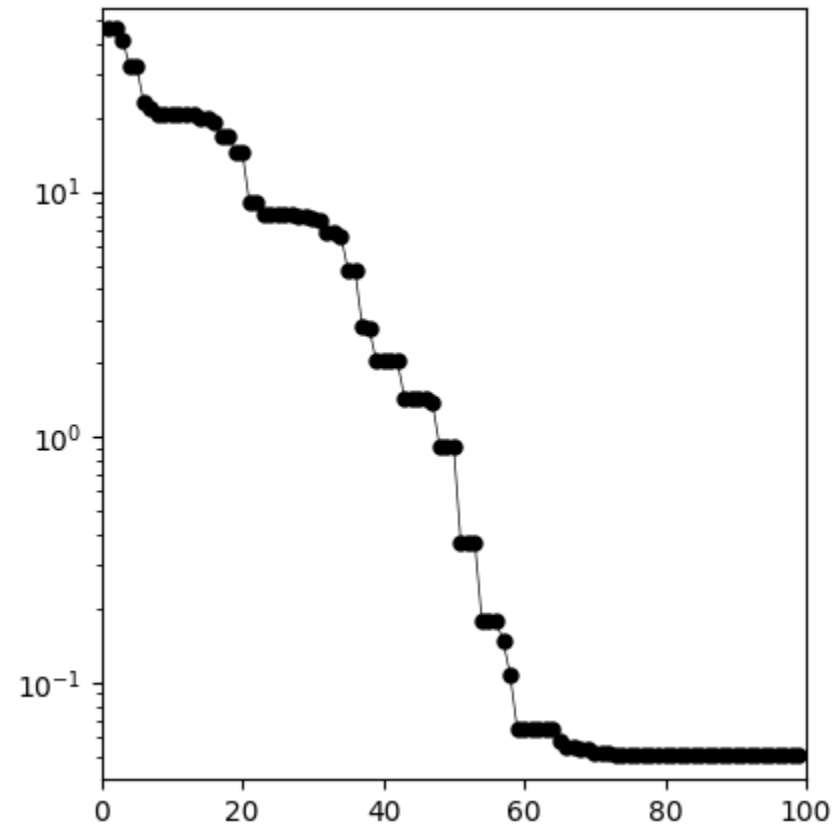
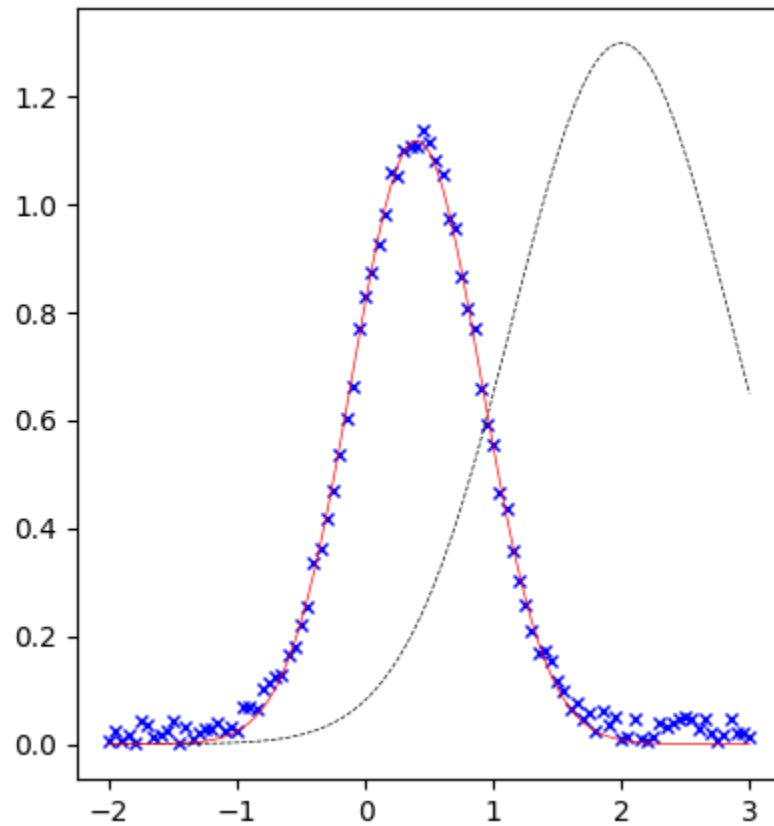
The result may be converged if w is wide enough to cover some region of the curve, even if the initial peak position is out of the FWHM of the target peak

Converged

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

method: nelder-mead (SIMPLEX)

initial values: $I0 = 1.3$, $x0 = 2.0$, $w = 1.0$



Marquardt 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

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

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

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

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

Levenberg-Marquardt method

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

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

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 $(n+1)$ vertices in n -dimension space

(単体: n 次元空間で $(n+1)$ 個の頂点を作る多面体)

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 $F(\mathbf{x}_i) > F(\mathbf{x}_{i'})$ ($i < i'$)

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

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 $\mathbf{x}_l - \mathbf{x}_G$ by the following selections

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

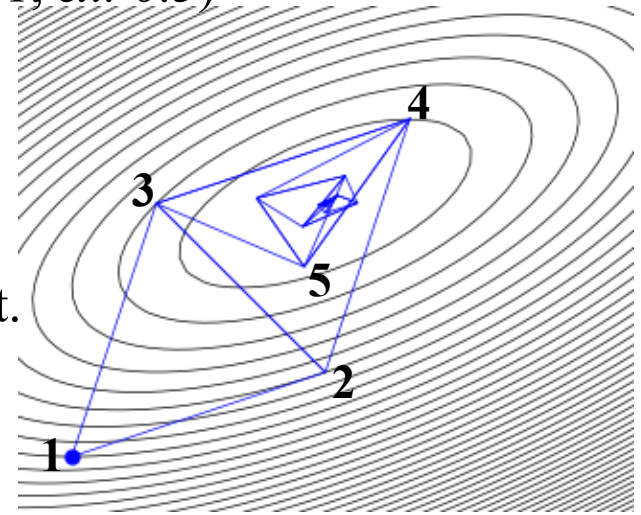
(ii) Expansion (拡大) : $\mathbf{x}_E = \gamma\mathbf{x}_R + (1 - \gamma)\mathbf{x}_G$ ($\gamma > 0$, ex. 2.0)

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

(iv) Reduction (縮小) : $\mathbf{x}_i \leftarrow \mathbf{x}_l + \sigma(\mathbf{x}_i - \mathbf{x}_l)$

4. The next simplex is chosen by comparing the trial points with the best, second-worst, and worst vertices, not simply by the first improvement over the worst point.

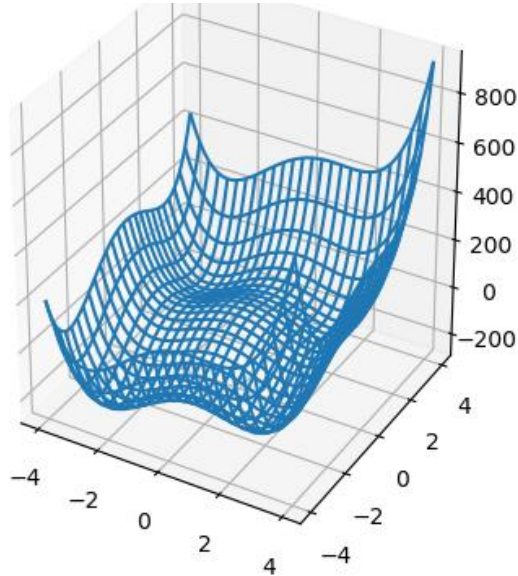
5. Repeat 2 - 4



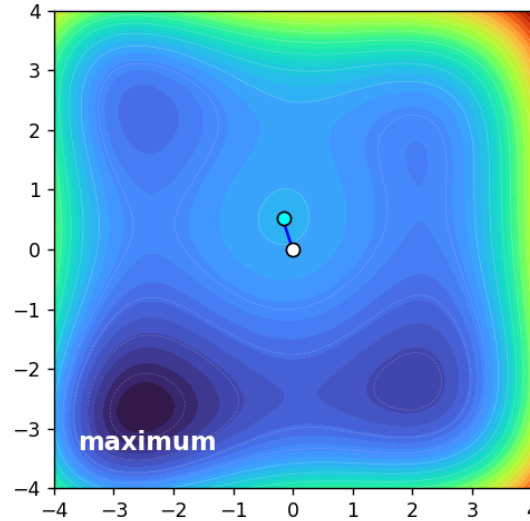
Newton, CG, vs SD

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

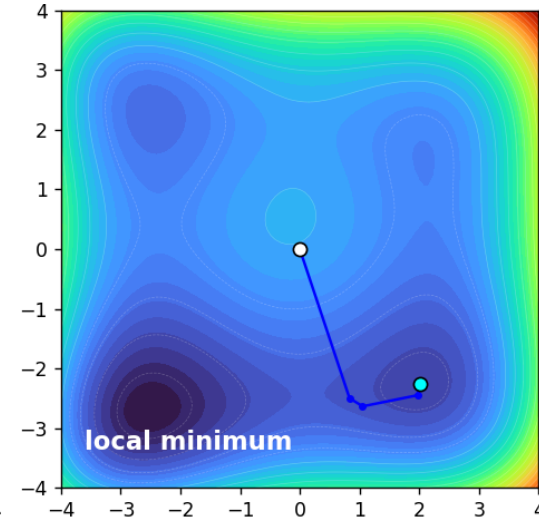
> `python optimize2d.py --mode [newton|cg|sd] --x0=x0 --y0=y0`



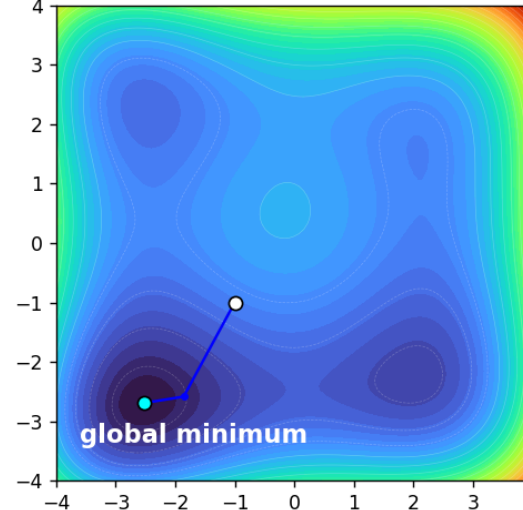
From (0.0 0.0) Newton



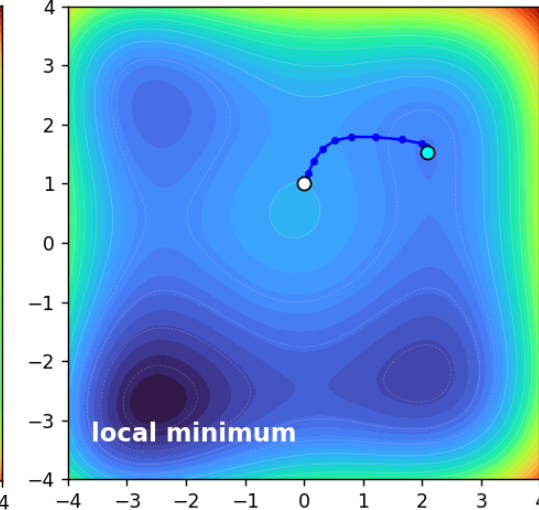
From (0.0 0.0) cg



From (-1.0 -1.0) cg



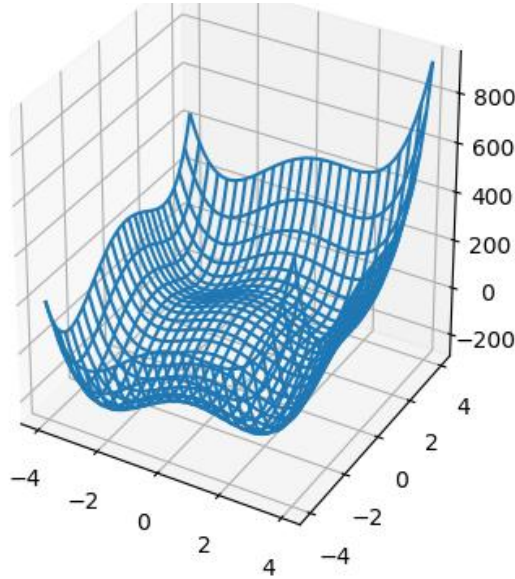
From (0.0 1.0) sd



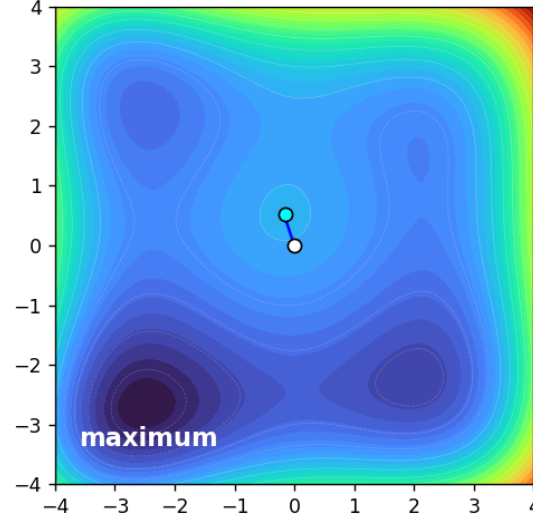
Newton, BFGS, vs. Simplex method

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

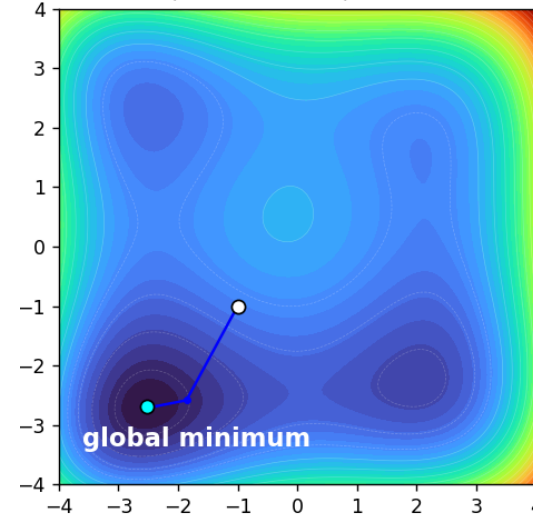
> `python optimize2d.py --method [newton|cg|sd|dfp|bfgs|simplex] --x0=x0 --y0=y0`



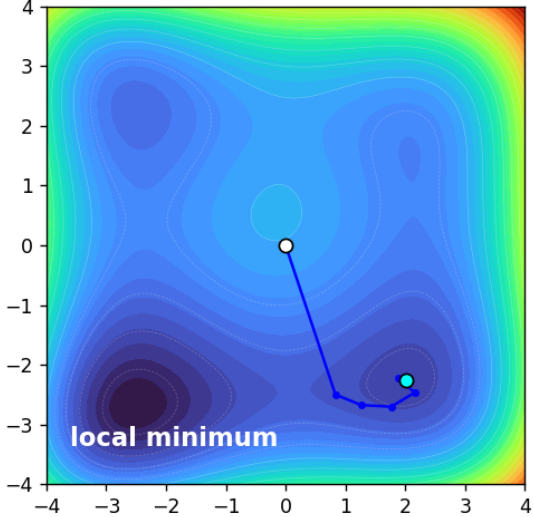
From (0.0 0.0) newton



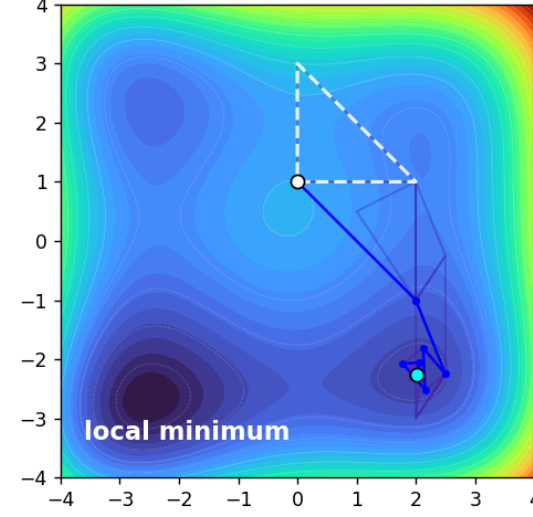
From (-1.0 -1.0) bfgs



From (0.0 0.0) bfgs



From (0.0 1.0) simplex



Main algorism:

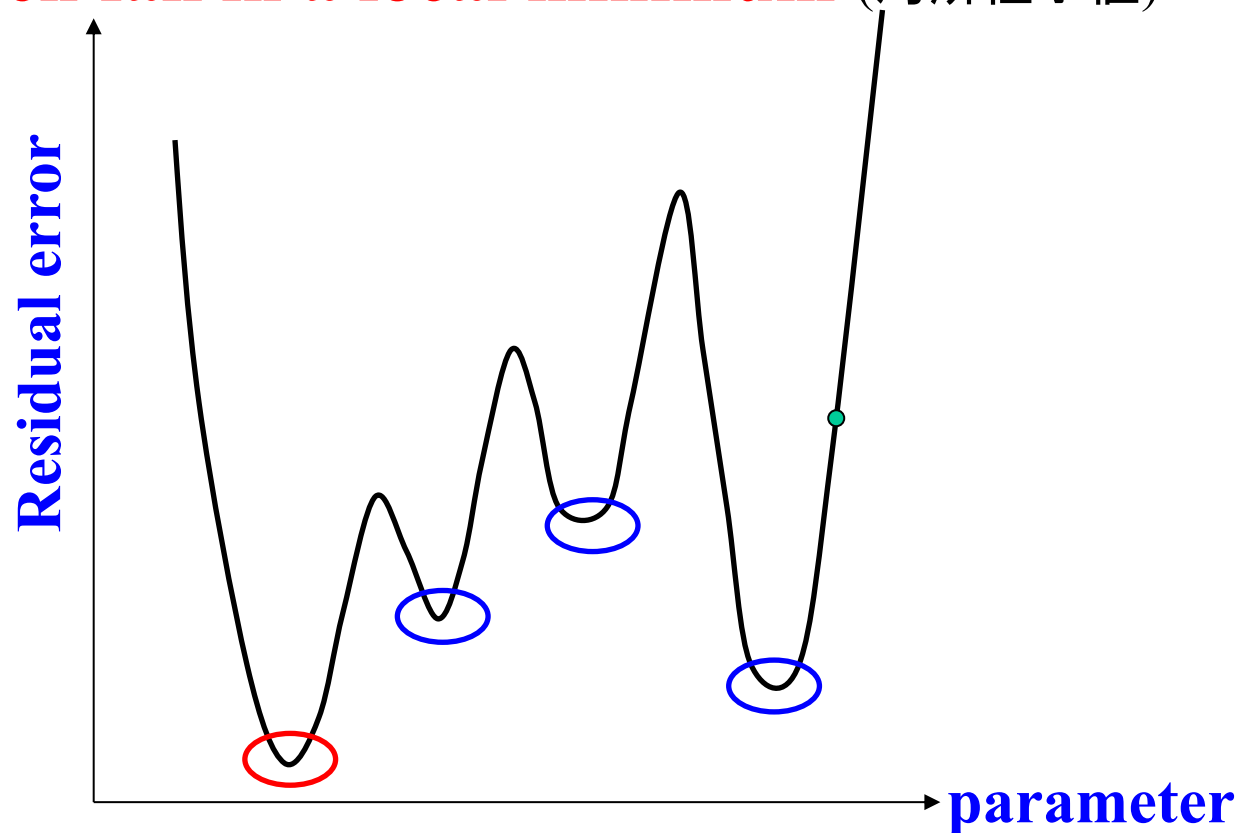
newton, sd, cg, broyden, dfp, bfgs
simplex

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

▪ **Davidon-Fletcher-Powell (DFP)**

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

B : Newton-Raphson method

Summary of nonlinear (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 derivatives are used to search minimum
Simple program, Slower convergence than NR and CG
- **Newton-Raphson method:**
Use 1st and 2nd derivatives (Hessian matrix) to efficiently find minimum
Fast convergence, easily diverged, complex program / May reach to a maximum
- **Quasi Newton method (準Newton法): DFP, BFGS, Broyden etc**
Use only 1st derivatives, approximating Hessian matrix
Better convergence by combining with linear search algorithms / Complex math
- **Conjugate Gradient method (共役勾配法):**
Use only 1st derivatives, search direction is corrected by conjugate gradient
Better convergence than NR, faster than SD / Complex math
- **Marquart method**
Only for least-squares fitting, Hessian matrix is build from 1st differentials
Simple program, slower convergence than NR

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 robust convergence

Appendix: python implementation

Tutorials (Japanese only now)

<http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/?page=tutorial>

source HTML | display HTML

- [学会等講演](#)
- [プレゼンテーション](#)
- [生成AI](#)
- [プログラミング](#)
- [数値解析](#)
- [結晶・回折・干渉](#)
- [結晶・回折・干渉\(内部限定資料\)](#)
- [物理・科学](#)
- [半導体デバイス](#)
- [データ科学](#)

▶ [pythonチュートリアル: Fortran/Cとの比較で学ぶpython文法](#)

▶ [pythonチュートリアル: Debug: 全体の流れ](#)

▶ [pythonチュートリアル: Debug - Tracebackメッセージの読み方](#)

▶ [pythonチュートリアル: Debug - pylint](#)

▶ [pythonチュートリアル: Debug - printデバッグ](#)

▶ [pythonチュートリアル: Debug - デバッグ](#)

▶ [Vibeコーディング \(難易度★\): van der Pauw法の形状補正因子を例に](#)

▶ [Vibeコーディング \(難易度★\): 非線形最小二乗法](#)

▶ [Vibeコーディング\(難易度★★★\): 散乱モデルによる移動度の温度依存性のフィッティング \(線形最小二乗法、非線形最小二乗法\)](#)

▶ [Vibeコーディング \(難易度★★★★\): 半導体のキャリア密度の温度依存性のフィッティング \(非線形最小二乗法\)](#)

▶ [Vibeコーディング \(難易度★★★★★\): 薄膜XRDの非線形最小二乗法 \(粒子群最適化\)](#)

フィッティングプログラムを作る流れ

1. データを準備
2. フィッティングさせる式と、フィッティングパラメータを明確にする
3. mode引数を用意し、mode=readで、データの読み込みと可視化を行い、正常に読み込めることを確認する
4. mode=simで、初期値のパラメータで計算した結果と入力データを比較するグラフを描き、計算関数が正常であることを確認する。また、初期値が離れていたら修正する
5. mode=fitでフィッティングを実行し、フィッティング前後と入力データを比較する。また、誤差を出力する

生成AIでフィッティングプログラムを作る: プロンプト例

Python で `scipy.optimize.minimize` を使った非線形最小二乗フィッティングプログラムを作成してください。

以下の仕様をすべて満たすこと。

【全体仕様】

- `scipy.optimize.minimize` を使用する
- `method` 引数をユーザーが選択できるようにする(例: Nelder-Mead, BFGS, Powell, CG, L-BFGS-B など)
- `argparse` を使う
- `mode` 引数を持ち、`mode=read / sim / fit` の 3 モードを実装する
- 入力ファイル名 `input`、最大繰り返し数 `nmaxiter`、収束判定条件 `tol`、フィッティングパラメータの初期値をコマンドライン引数で与えられるようにする
- `repeat` 出力と途中経過を標準出力に表示する
- `matplotlib` を用いて可視化する

【データ】

- `input` で与えられる CSV または Excelファイル を読み込む
- 1 列目を `x`、2 列目を `y` として扱う

【フィッティング関数】

- プロンプト内でユーザーが任意の関数 `f(x, params)` を指定できるようにする
- 例: `f(x, a, b, c) = a * exp(-b * x) + c`
- 関数はプログラム内に明示的に定義する
- 残差平方和 $RSS = \sum (f(x_i, params) - y_i)^2$ を `minimize` の目的関数とする

【mode=read】

- データを読み込み、`x` と `y` の散布図を描く
- データが正常に読み込めていることを確認できるようにする

【mode=sim】

- 初期値パラメータで `f(x)` を計算し、入力データと比較するグラフを描く
- 計算関数が正常に動作していることを確認する
- 初期値が明らかに不適切な場合は警告を出す

【mode=fit】

- `minimize` を用いて非線形最小二乗フィッティングを行う
- `method` は `argparse` の引数で指定できるようにする
- フィッティング前後の曲線と入力データを比較するグラフを描く
- フィッティング後のパラメータ、誤差、繰り返し回数を出力する
- `minimize` の `callback` を使って `iteration` ごとの途中経過(パラメータ・RSS)を標準出力に表示する

【その他】

- コードは読みやすく、関数化して整理する
- `main()` を用意し、`if __name__ == "__main__":` で実行する構造にする

生成例: `fit_generated.py`
`fit_generated.csv`

非線形最小二乗法のライブラリ・関数

Pythonで準標準ライブラリを使う非線形回帰は、いくつかの選択肢があります

1. `scipy.optimize.curve_fit()`

プログラム例: `peakfit05.py` ([フィッティング前後の可視化](#))

使い方: 最適化パラメータのリスト, 共分散行列

`= curve_fit(モデル関数, xデータ, yデータ, p0=初期値のリスト)`

特長: 簡単に使える (モデル関数を定義するだけでOK)

短所: 最適化途中の情報を得られない

2. `scipy.optimize.minimize()`

プログラム例: `peakfit10.py` ([アニメーション機能](#))

使い方: `result = minimize(最小化関数, 初期値のリスト, method=アルゴリズム, callback=callback関数)`

最適化パラメータ: `result.x`

特長: `callback`関数を使うことで、最適化途中の情報を得られる

最適化アルゴリズムを選択できる

短所: 最小化関数を定義しないといけない

最適化アルゴリズムを選択しないといけない (不明な場合は `nelder-mead` を使う)

scipy.optimize.minimize()で使えるアルゴリズム

scipy.optimize.minimize()では、様々な最適化アルゴリズム (method) を使えます。
以下は、データの非線形最小二乗法でよく使われるものです

アルゴリズム	Method	広域安定性 (初期値の寛容性)	収束速度	プログラムの複雑さ
Newton法	実装されていない	× 高い精度の初期値が必要 (ほぼまともな解は得られない)	◎ ほぼ最速	× 最小化関数の2次微分関数が必要
Nelder-Mead (単体法)	nelder-mead	◎ この中では最も寛容	× とにかく遅い 特に変数が多 くなると無理	◎ 最小化関数だけでOK
BFGS / L-BFGS-B (準Newton法)	bfgs	○	○	○ 最小化関数の1次微分関数が必要 (jac=None で自動計算)
Conjugate gradient (共役勾配法)	cg	○	○	○ 最小化関数の1次微分関数が必要 (jac=None で自動計算)

収束計算を安定させる方法

1. フィッティングパラメータ a に制約範囲 $[amin, amax]$ を課す。
 - (i) **境界付き最適化** (L-BFGS-B / trust-constr / least_squares) を使う
 - (ii) 範囲から外れた場合は 最小化関数に **penalty** を足す
$$\text{penalty} = k(a - amin)^2 \text{ if } a < amin, \quad k(a - amax)^2 \text{ if } a > amax$$
勾配ベースの最適化を使う場合は、 $\text{softplus}(x) = \ln(1 + e^x) \approx \max(0, x)$ を使う方が良い
$$\text{penalty} = k\{\text{softplus}(amin-a)\}^2 + k\{\text{softplus}(a-amax)\}^2$$
 - a が制約範囲の境界に張り付く場合や、範囲外に出やすい場合は、 a の項が必要か / 適切か、妥当性を再検討する
 - スケール変換した場合は、誤差伝播法によりもとのパラメータの誤差を計算する
2. フィッティングパラメータが正値で、桁が大きく異なるものが含まれる場合:
 - a の対数 $\ln a = \ln(a)$ を最適化する
 - $a > 0$ が保証される
 - 桁が大きく異なる フィッティングパラメータでも、各パラメータの重みをほぼ等しくフィッティングできる
 - a は 0 にはならないが、十分小さければ 0 と判断できる。定量的には、寄与度で判断する

非線形最適化法の比較

最適化の分類

- 勾配法: SD, CG, BFGS, etc
収束速度早い、
広域収束性が保証されない、大域最適解を得るのが難しい
- 直接探索法・集団探索法: Simplex (nelder-mead), 遺伝的アルゴリズム, etc
収束速度遅い、
広域収束性が良い、大域最適解を得られる可能性が高い

比較:

- nelder-mead (simplex)
- 遺伝的アルゴリズム (Genetic algorithm: GA)
- レプリカ交換モンテカルロ法 (Replica exchange MC: REMC)
- 粒子群最適化 (Particle swarm optimization: PSO)
- **焼きなまし法 (Simulated annealing: SA)**

遺伝的アルゴリズム (GA: Genetic Algorithm)

- 生物進化のしくみ(選択・交叉・突然変異)を模倣した最適化法
⇒ **集団進化により大域解を探索**
- 勾配を必要としない
- 離散問題、組合せ最適化、多峰性問題に強い
- 計算コストが大きくなりやすい

- 解候補を 個体、個体の集合を 集団 (population) とよぶ
- 各個体の適応度 ($f(x)$) を評価し、適応度の高い個体を優先して残す

アルゴリズム

1. 初期集団 $\{x_i\}$ をランダムに生成
2. 各個体の適応度 $f(x_i)$ を評価
3. 適応度に基づいて親個体を選択(**選択: selection**)
4. 交叉・突然変異により子個体を生成
 - 交叉 (crossover)** : 親個体の情報を組み合わせて子個体を作る
 - 突然変異 (mutation)** : 一部をランダムに変えて多様性を保つ
5. 子個体を中心に新しい集団に置き換え (**世代交代: replacement**)、上記を繰り返す

レプリカ交換モンテカルロ法

永田賢二 他、交換モンテカルロ法による変数選択問題における解の効率的な全数検索、日本人工知能学会第28回年会 (2014) G2-3

- 乱数を使って状態空間をサンプリング \Rightarrow 局所解に落ち込む可能性がある
- 高温レプリカ: 障壁を超えて局所解から脱出する
- 低温レプリカ: 低エネルギー状態を精密に探索する
- 状態交換により、低温側に高温側の探索成果を伝える \Rightarrow 大域解の探索へ
- 分布関数を再現するサンプリング
 \Rightarrow 期待値、状態密度、自由エネルギーの計算が可能

1. 近いが異なる逆温度 β_m で、複数のボルツマン分布に従う分布 $p(S_m; \beta_m)$ をモンテカルロ法で作る

2. 隣り合う温度間で状態を交換し、

$$v = \frac{p(S_{m+1}; \beta_m) p(S_m; \beta_{m+1})}{p(S_m; \beta_m) p(S_{m+1}; \beta_{m+1})} = \exp[(\beta_{m+1} - \beta_m)(E(S_{m+1}) - E(S_m))]$$

$$p(S_m \leftrightarrow S_{m+1}) = \min(1, v)$$

の確率で交換を実施する

粒子群最適化 (PSO: Particle swarm optimization)

- 鳥の群れや魚群が餌の位置を見つける行動をモデル化
- 勾配を必要としない
- 非線形性、多峰性、ノイズのある最適化に強い
- 解候補を粒子、粒子の集合を群れ (swarm) とよぶ
- 粒子は探索空間中の位置 (x_i) と速度 (v_i) を持つ

- 粒子は次の 2つの情報を参考に移動して最適解を探索
 1. 自分が過去に見つけた最良位置
 2. 群れ全体の最良位
- 速度更新:
$$v_i^{(t+1)} = w v_i^{(t)} + c_1 r_1 (p_i - x_i^{(t)}) + c_2 r_2 (g - x_i^{(t)})$$

慣性項 自分の成功体験へ戻る 群れの成功へ向かう

w: 慣性係数, p_i : 粒子の個人最良位置, g: 全体最良位置
(r_1, r_2): 0~1の乱数, (c_1, c_2): 学習係数
- 位置更新: $x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$
- 上記で見つけた最適位置を更新しながら繰り返す

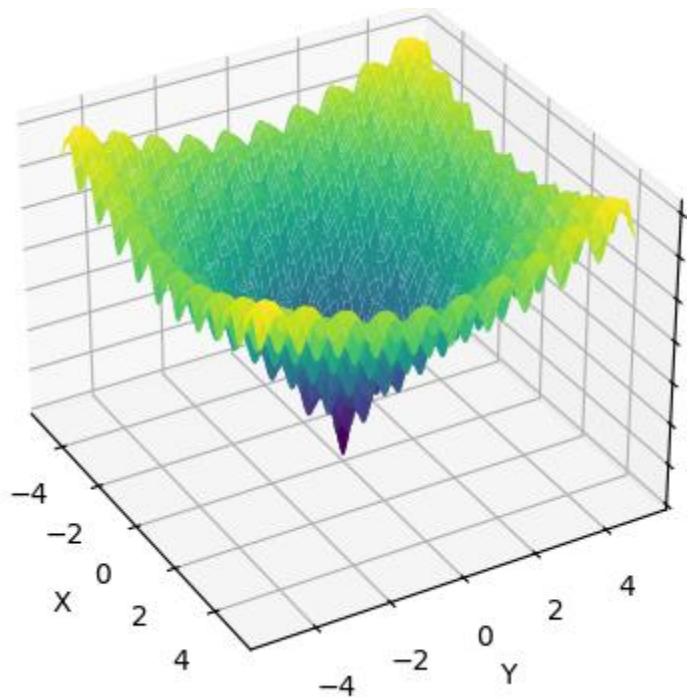
複数候補による最適化法の比較

アルゴリズム	候補数	確率性	局所脱出	計算コスト
Simplex	1集合	なし	弱い	小
PSO	多数	あり	中	中
GA	多数	強い	強	大
REMC	多数	温度	強	中～大

単純実装したプログラムでの比較

Python optimize_ga_swarm_remc.py: simplex, ga, pso, remc

Ackley関数の最小化: $f = 0$ @ (0, 0)



初期範囲 [-30, 30] / simplex: (5,5)~[-30,30]

of T (remc): 10

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

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

比較の結果

- Simplex法は勾配法と比較すると遅いが、**初期値が近い場合はこの中では最速**
- **PSOは次点(複数候補を探索するため初期値が遠くても収束しやすい)**
- GA, REMCは有効サンプルの割合が低く、関数評価回数が多い
- SAを含め、GA, REMCでは収束を早める工夫が必要
次の状態を作る際、勾配 × 乱数 にするなど

注意:

- Simplex法のみ、初期値を与えている(局所探索法)
- REMCではレプリカ数(温度数)が多いため計算コストが大きい
- GA, REMCでは乱数範囲のチューニングが必要

**薄膜XRDフリンジフィッティングでは局所解が多いため、
初期値依存の小さいPSOを採用した**