2018 年 1 月 22 日作成

名古屋工業大学 先進セラミックス研究センター　井田 隆

2022 年 1 月 18 日更新

第３部

最適化とモンテカルロ法

８．最適化　Optimization

最適化（最適化計算） optimization とは，（コンピュータを使って）函数(かんすう)の**最大**あるいは**最小**を求めようとすることです（補足 8.A）。**最適化**は，技術・産業・政治・経済・教育などの問題を解決するために重要です。**機械学習** macnine learning では，解答例つきの訓練データ（例題）に対する反応のしかたがなるべく正解に近くなるように，機械の挙動を決めるパラメータが最適化されます。

今までに学んだこと（第２部　構造のシミュレーション，ポテンシャルの計算など）と**最適化**を組み合わせることで，例えば「物質の安定な構造」を推定することができます。ポテンシャルを最小にする構造が「エネルギー的に最も安定な構造」と呼ばれることがあります。

一定の圧力のもとで最も安定な構造は，**エンタルピー** enthalpy

$$H = U + PV \tag{8.1}$$

を最小にする構造として求められます。ここで U は**内部エネルギー** internal engergy，P は**圧力** pressure，V は**体積** volume です。周期的境界条件を用いる場合には，U として「基本セル当たりの内部エネルギー」，V として「基本セルの体積」を用い，「基本セルあたりのエンタルピー」を最適化することになります。

一定温度・一定圧力で最も安定な構造は**ギブス自由エネルギー** Gibbs free energy

$$G = H - TS = U + PV - TS \tag{8.2}$$

を最小にする構造になるはずです。ここで T は**温度** temperature，S は**エントロピー** entropy です。ただし，有限温度でのダイナミックスのシミュレーションのためには，実際には運動方程式（あるいはハミルトンの発展方程式）を解く方法（分子動力学計算 molecular dynamics calculation；**MD 法**) が多く使われます。

周期的境界条件を用いると，一般的にはエントロピーが過小に評価されます。

最適化は，コンピュータを使った構造シミュレーションだけではなく，実験データ（力学的／光学的／電気的／磁気的な測定の結果）を解析する場合にも用いられます。

実験 experiment・**観測** observation で得られたデータに，パラメータを含む理論モデルをあてはめて，ずれを最小にするようなパラメータを求めるような最適化のしかたは**曲線当て嵌め** curve fitting とも呼ばれ，実験データの解析に古くから用いられてきました。ほぼ同じことが**回帰** regression と呼ばれることもあります（補足 8.B）。

実験データを解析する場合の基本的な考え方として，**ベイズ推定**，**最尤推定**，**最小自乗推定**があります。

ベイズ推定 Bayesian inference は「条件付きの確率」を求めることと同じで「観測されたデータ（**エビデンス** evidence；周辺尤度 marginal likelihood）に基づいて，仮説 hypothesis・**事前確率** prior probability・先入観・主観 (subject) を修正した**事後確率** posterior probability を求める」方法をとります。この過程で，一般的には**尤度函数** likelihood estimator が必要となります。尤度函数と逆の意味を持つ**損失函数** loss function を最小化するという表現の取られる場合も多いようです。**ベイズ型の機械学習**では，独立な訓練データを使って逐次的に機械に学習させることができます。

最尤推定は「観測されたデータの出現する確率が最大となるようなモデルを求める」ことともに「**一様事前分布**を仮定した**ベイズ推定事後確率**の最大を求める」こととも同じことです。構造モデルなどの理論モデルによって，実験データの出現する確率を函数の形（理論 theory・尤度函数 likelihood esitimator) で表すことができれば，**最尤推定**による「実験データの解釈」は，**尤度函数**の最大を求める問題と同じことです。

例えば，

安定構造のシミュレーション　＝　ポテンシャル函数の最小化

最尤推定による実験データ解析　＝　尤度函数（データの出現確率）の最大化

のような関係があります。

最小自乗推定は，一般的には観測値 $\{y_1, \cdots, y_n\}_{\rm obs}$ と計算値（予測値）$\{y_1, \cdots, y_n\}_{\rm cal}$ の差を誤差 $\{\Delta y_1, \cdots, \Delta y_n\}$で除した値の自乗和：

$$S = \sum_{j=1}^{n} \frac{[(y_j)_{\rm obs} - (y_j)_{\rm cal}]^2}{(\Delta y_j)^2} \tag{8.3}$$

を最小化する方法です。観測値の統計分布が 標準偏差 $\{\Delta y_1, \cdots, \Delta y_n\}$ の正規分布に従う場合，最小自乗推定は最尤推定と同じことになります。

任意の n 変数函数 $f(x_1, \cdots, x_n)$ の**最大化**は，函数 $g(x_1, \cdots, x_n) = -f(x_1, \cdots, x_n)$ の**最小化**と同じことです。過去の歴史では「**最適化** optimization」は「函数の最小を求める方法」という文脈で用いられることが多かったのですが，最近では「確率を現す函数の最大を求め

る方法」「エントロピーを表す函数を最大にする方法」という文脈で用いる場合の増す傾向はあります。

8－1　最適化の方法　Methods for optimization

コンピュータを使った最適化の代表的な方法に以下のようなものがあります。

（1）グリッド・サーチ法

（2）黄金分割法（一次元の最小化）とそれを使った多次元の最小化の方法

（3）ネルダー・ミード法（滑降シンプレックス法）

（4）ニュートン法と関連する方法

（5）モンテカルロ法と関連する方法（擬似乱数を使う方法）

グリッド・サーチ法 grid-search method（網探索法）とは，函数の変数が取りうる範囲を細かく区切り，すべての取りうる変数値の組み合わせで函数値を評価し，最小の値をとる組み合わせを選ぶ方法です。計算の効率は良くなく，得られる結果の精度も高くなりませんが，最も安全で確実な方法とも言えます。他の方法で最小を求めるとしても，「正しく最小が得られているかを確認する」などの目的で有効です。**グリッド・サーチ法**は「最適化アルゴリズム」に分類されないかもしれませんが，局所的な最適化アルゴリズムと組み合わせて，大域的な最適化の目的で用いられる場合はあります。

黄金分割法については8－2節で，**ネルダー・ミード法**（滑降シンプレックス法）については8－3節で紹介します。

ニュートン法 Newton's method による最適化は，基本的には「微分がゼロになるような解」を求める方法で，最適解から離れた位置では，効率が悪かったり挙動が不安定になる一方で，最適解の近くでは多くの場合に効率良く働きます。他の方法と組み合わせて用いられるのも普通です。**準ニュートン法** quasi-Newton method と呼ばれる方法も使われます（補足 8.1.A）。準ニュートン法に分類される方法のうち，ブロイデン・フレッチャー・ゴールドファープ・シャンノ法 (Broyden-Fletcher-Goldfarb-Shanno method)**（BFGS 法）**が比較的良く使われます。

黄金分割法，ネルダー・ミード法，準ニュートン法は，いずれも**極小** local minimum を求めるための方法であり，局所的な最適化のための算法 (algorithm for local optimization) に分類されます。多くの「**局所的な最適化アルゴリズム**」が知られています。

大域的な最小 global minimum を求めるための算法，「**大域的な最適化アルゴリズム**」(algorithm for global optimization) の多くに**モンテカルロ法** Monte Carlo method が組み込まれます。ただし「大域的な最適化アルゴリズム」を使ったからと言っても，必ず大域的な最適化が実現されるというわけではありません。

モンテカルロ法とモンテカルロ法に関連する方法，「大域的な最適化アルゴリズム」については9章で紹介します。

8－2　黄金分割法　Golden section method

$$\phi = \frac{\sqrt{5}+1}{2} = 1.618\,033\cdots \tag{8.2.1}$$

あるいは

$$r = \frac{1}{\phi} = \frac{2}{1+\sqrt{5}} = \frac{\sqrt{5}-1}{2} = 0.618\,033\cdots \tag{8.2.2}$$

が**黄金比** golden ratio と呼ばれます。ϕ の値のことが黄金数 golden number と呼ばれることもあります。
長辺と短辺の比が黄金比となる長方形が黄金長方形と呼ばれます（補足 8.2.A）。黄金比はフィボナッチ数列 Fibonacci 数列の隣り合う項の比の極限値としても知られています（補足 8.2.B）。
「自然や人工の造形，芸術作品に黄金比が隠されていることが多い」という人もいますが，「それほどでもない」という人も少なくありません（補足 8.2.C）。

区間を$1-r : r \approx 0.382 : 0.618$ あるいは $r : 1-r \approx 0.618 : 0.382$ に分割することを黄金分割と呼びます。

黄金分割法は 1 次元の最小化アルゴリズムであり，比較的安定に動作し，プログラミングも容易です。以下のような手続きで，有限の区間内での函数の最小を求めることができます（補足 8.2.D）。

【黄金分割アルゴリズム】

(i)　区間の下限を a, 上限を d とする。$y_1 = f(a), y_4 = f(d)$ の値を計算する。

(ii)　以下の式に従って，区間を黄金分割する二つの点 b，c をとる。

$$b = ra + (1-r)\,d \tag{8.2.3}$$

$$c = (1-r)\,a + rd \tag{8.2.4}$$

b は区間 $[a,d]$ を $1-r : r$ の比に内分する点であり，c は区間 $[a,d]$ を $r : 1-r$ の比に内分する点である。

$y_2 = f(b)$，　$y_3 = f(c)$ の値を計算する。

(iii)　もし $y_2 = f(b) < y_3 = f(c)$ なら，最小となるのは $[a,c]$ の区間内だから，以下の代入：$d \leftarrow c$，$c \leftarrow b$，$y_3 = f(c) \leftarrow y_2 = f(b)$ を行い，$b = ra + (1-r)\,d$ として $y_2 = f(b)$ の値を計算する。 $y_3 = f(c) \le y_2 = f(b)$ なら，最小となるのは $[b,d]$ の区間

内なので，以下の代入：$a \leftarrow b$，$b \leftarrow c$，$y_2 = f(b) \leftarrow y_3 = f(c)$ を行い，
$c = (1-r)a + rd$ として $y_3 = f(c)$ の値を計算する。

(iv) (iii) に戻る。

たとえば，Fig. 8.2.2 に示すように，函数：

$$f(x) = -\exp\left[-(x-0.5)^4\right] - \exp\left[-(x-1.5)^2\right] \tag{8.2.5}$$

について，$x \in [0, 2]$ として**黄金分割探索** golden section search を繰り返せば，正しい極小位置 ($x = 1.088\,43$) が求められます。

黄金分割法は「局所的な最適化法」に分類されるものであり，正しい最小 global minimum でない極小 local minima（偽の最小 false minima）に陥る場合があることには注意すべきです（補足 8.2.E）。

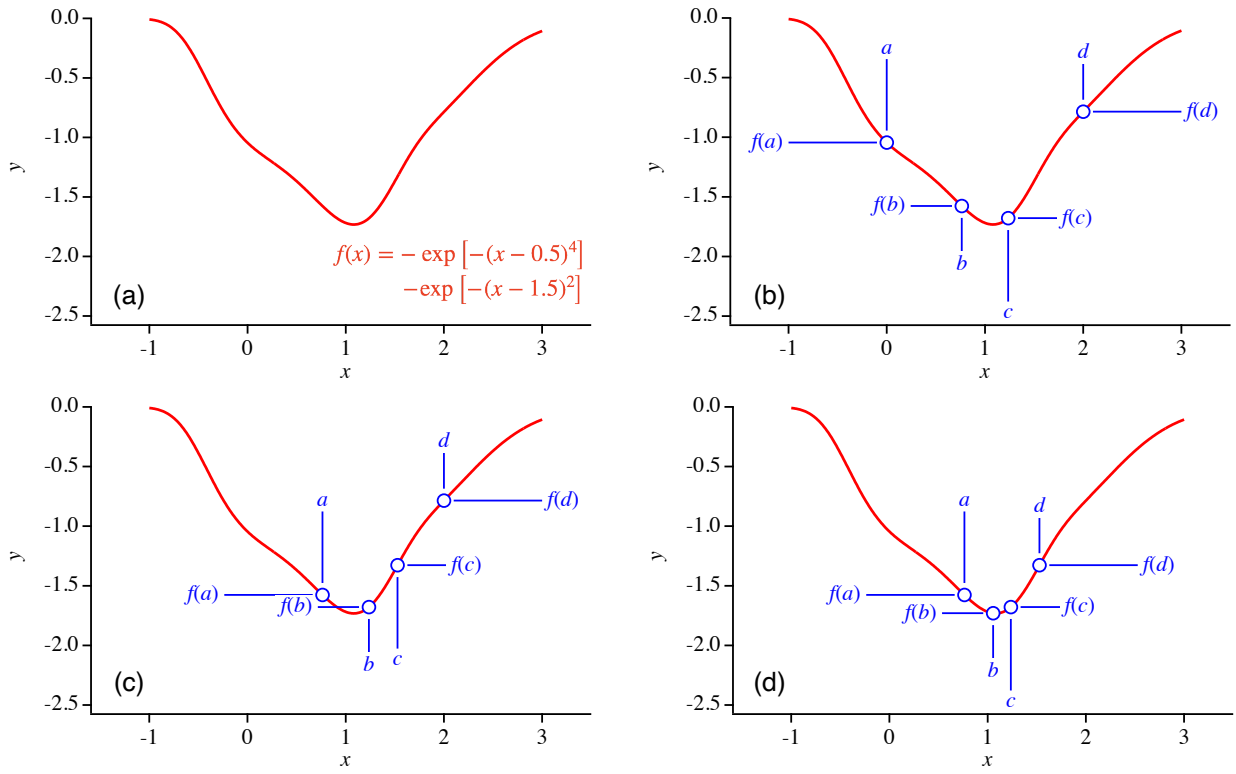

Fig. 8.2.2　黄金分割法を函数 $f(x) = -\exp\left[-0.2\left(x-0.5\right)^4\right] - \exp\left[-0.2\left(x-1.5\right)^2\right]$ の最小化に適用する場合の計算の進行例。初期区間を $x \in [0, 2]$ とする。(a) 函数の形状。(b) はじめの標本点配置と函数値。$f(b) > f(c)$ なので標本点 a と函数値 $f(a)$ を捨てて，$a \leftarrow b$，$f(a) \leftarrow f(b)$，$b \leftarrow c$，$f(b) \leftarrow f(c)$ とする。d と $f(d)$ の値はそのまま維持する。(c) 次の段階での標本点配置と函数値。新しい標本点として，区間 $[a, d]$ を黄金分割する点として c の位置を求め，函数値 $f(c)$ を新しく計算する。この結果 $f(b) < f(c)$ となったので，ここでは，標本点 d と函数値 $f(d)$ を捨てて，$d \leftarrow c$，$f(d) \leftarrow f(c)$，$c \leftarrow b$，$f(c) \leftarrow f(b)$ とする。a と $f(a)$ の値はそのまま維持する。(d) 次の段

階での標本点配置と函数値。新しい標本点として，区間 $[a,d]$ を黄金分割する点として c の位置を求め，函数値 $f(c)$ を新しく計算する。

黄金分割法では，最適解に近づいてから収束するまでにかなり多くの計算ステップを必要とするので，一変数函数の局所的な最適化のためには「黄金分割法」と「一変数函数の微分についてのニュートン・ラフソン法」とも言える「放物線補間」(parabolic interpolation) とを組み合わせて収束を早くさせる**ブレントの方法** Brent's method と呼ばれる方法の用いられるのも普通です（補足 8.2.F）。

8－3　ネルダー・ミード法　Nelder-Mead method

滑降シンプレックス法 downhill simplex method とも**超多面体法** polytope method，**アメーバ法** amoeba method などとも呼ばれます。二つ以上の変数を持つ函数の最適化で用いられます。この手法を自力でコーディングするには少し手間がかかりますが，動作は比較的安定・確実なので，知っておくと良い方法です。

黄金分割法が一次元で「解を含む範囲を挟みうちしながら縮小していく」のと同じように，ネルダー・ミード法では「解を含む空間を囲む多面体（超多面体）（**シンプレックス** simplex）を縮小していく」ような操作を繰り返します。ただし，厳密な意味で解を囲む状態を保つでなく，**シンプレックス**の頂点について (1) **反転** inversion あるいは (2) **反転拡大** inversion & expansion，(3) **収縮** contraction，(4) **縮小** reduction のいずれかの操作をほどこすことを繰り返して，**シンプレックス**を変形・移動させながら，極小位置の近くで徐々にシンプレックスを小さくさせる方法をとります。

ネルダー・ミード法では，m 個のパラメータ（変数）を含む函数（m 次元函数）を最適化するためには，m 次元の空間（超空間）で $(m+1)$ 個の頂点を持つ**シンプレックス**（多角形あるいは多面体，超多面体）を変形・移動する操作を行います。２変数の函数であれば，平面の上で三角形を使います。３変数の函数なら三次元空間で四面体を使います。４変数以上の函数であれば４次元超空間で４次元超多面体を使います。
以下に，**ネルダー・ミード法**で極小を求める具体的な作業を示します。

【ネルダー・ミード法のアルゴリズム】

（１）**シンプレックス**の**頂点** vertex の座標を $\mathbf{p}_i$ $(i=1,\cdots,m+1)$ とします。はじめに頂点の「初期配置」を決めなければいけません。事実上この初期配置の決め方で最適化がうまくいくかが決まります。初期配置はユーザに指定させれば良いのですが，m $(m+1)$ 個の数値を指定させることになり，このことでネルダー・ミード法が扱いづらいものになる傾向があります。

乱数を使ってランダムな初期配置から始める場合もあります。

ユーザーに「粗い推定値」 $\tilde{\mathbf{p}} = \left(\tilde{p}_1, \cdots, \tilde{p}_m\right)$ と「粗い推定誤差」$\Delta\tilde{\mathbf{p}} = \left(\Delta\tilde{p}_1, \cdots, \Delta\tilde{p}_m\right)$ の $2m$ 個の数値のみ指定させ，$(m+1)$ 個の頂点を $\tilde{\mathbf{p}}_1 = \left(\tilde{p}_1 + \Delta\tilde{p}_1, \tilde{p}_2, \cdots, \tilde{p}_m\right)$，$\tilde{\mathbf{p}}_2 = \left(\tilde{p}_1, \tilde{p}_2 + \Delta\tilde{p}_2, \cdots, \tilde{p}_m\right)$，$\cdots$，$\tilde{\mathbf{p}}_m = \left(\tilde{p}_1, \tilde{p}_2, \cdots, \tilde{p}_m + \Delta\tilde{p}_m\right)$，$\tilde{\mathbf{p}}_{m+1} = \left(\tilde{p}_1, \cdots, \tilde{p}_m\right)$ のように配置するのも現実的な対応のしかたです。このことは，二変数函数の最適化に用いる場合には，初期配置の超多面体（二次元なら三角形）を Fig. 8.3.1 のような直角三角形に設定することと同じことです。

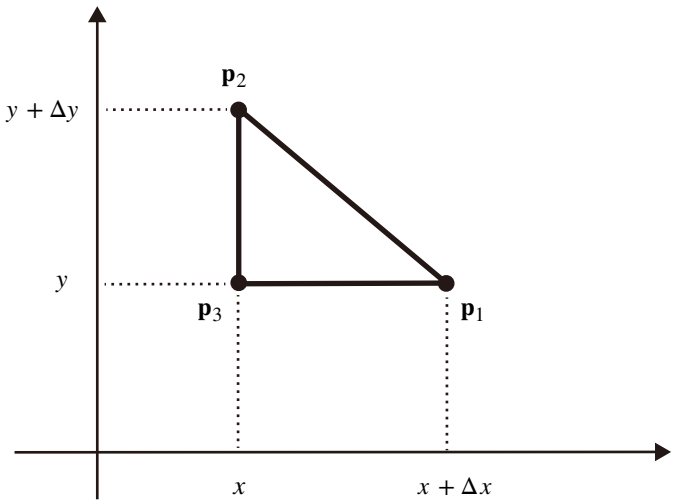

Fig. 8.3.1　初期シンプレックスの例。ユーザーに粗い推定値 (x, y) と粗い推定誤差 $(\Delta x, \Delta y)$ を指定させる場合。

（２）各ステップ毎に，頂点を函数値に応じて序列化 (sorting) します。特に**最悪点** worst point と**第二最悪点** second worst point，**最良点** best point の３点の序列を決める必要があります。ここでは，単純化のために

$$f(\mathbf{p}_1) \le f(\mathbf{p}_2) \le \cdots \le f(\mathbf{p}_m) \le f(\mathbf{p}_{m+1})$$

の関係が成り立つように，すべての頂点を序列化するとします。**最良点**は $\mathbf{p}_1$，**第二最悪点**は $\mathbf{p}_m$，**最悪点**は $\mathbf{p}_{m+1}$ となります。二変数の場合には頂点が３つの三角形を用います。二変数函数を最適化の対象とする場合には，たとえば Fig.8.3.2 に示すようになるとします。

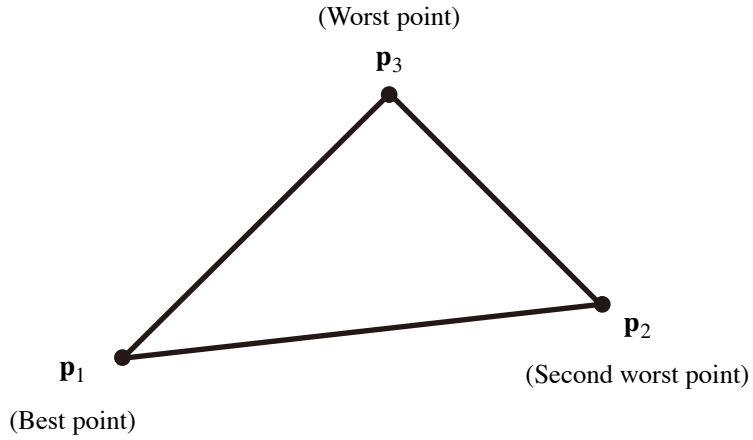

Fig. 8.3.2　頂点の序列化

以下の手順を繰り返します。

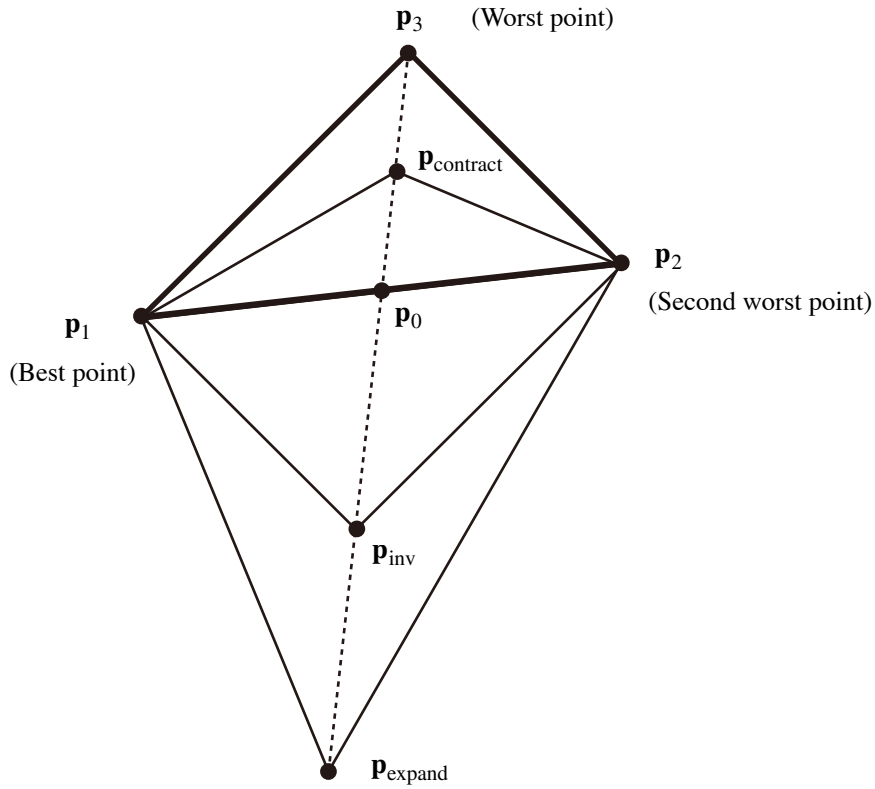

Fig. 8.3.3　Fig. 8.3.2 のように序列化された場合の最悪点 $\mathbf{p}_3$と，最悪点以外の重心 $\mathbf{p}_0$, 反転点 $\mathbf{p}_{\mathrm{inv}}$, 反転拡大点 $\mathbf{p}_{\mathrm{expand}}$, 収縮点 $\mathbf{p}_{\mathrm{contract}}$ の関係

（3）**最悪点** $\mathbf{p}_{m+1}$ 以外のすべての点 $\{\mathbf{p}_1, \cdots, \mathbf{p}_m\}$ の**重心** gravity center：

$$\mathbf{p}_0 = \frac{1}{m}\sum_{i=1}^{m} \mathbf{p}_i$$

を求めます。二変数函数の場合の**重心**の位置は，Fig.8.3.3 に示すようになります。

（４）**最悪点** $\mathbf{p}_{m+1}$ を**重心** $\mathbf{p}_0$ に対して**反転** (inversion) した位置（**反転点**） $\mathbf{p}_{\text{inv}}$ を求めます 。**反転点** $\mathbf{p}_{\text{inv}}$ は，以下の式で計算できます。

$$\mathbf{p}_{\text{inv}} = \mathbf{p}_0 + \left(\mathbf{p}_0 - \mathbf{p}_{m+1}\right) = 2\mathbf{p}_0 - \mathbf{p}_{m+1}$$

また**反転点** $\mathbf{p}_{\text{inv}}$ での函数の値 $f(\mathbf{p}_{\text{inv}})$ を求めます。二変数函数の場合の**反転点**の位置は，Fig.8.3.3 に示すようになります。

（５）**反転点** $\mathbf{p}_{\text{inv}}$ が今までの**最良点** $\mathbf{p}_1$ よりは悪い（$f(\mathbf{p}_1) \le f(\mathbf{p}_{\text{inv}})$）が，**第二最悪点** $\mathbf{p}_m$ より良い場合（$f(\mathbf{p}_{\text{inv}}) \le f(\mathbf{p}_m)$），つまり

$$f(\mathbf{p}_1) \le f(\mathbf{p}_{\text{inv}}) \le f(\mathbf{p}_m)$$

の関係があるとき，今までの**最悪点** $\mathbf{p}_{m+1}$ を捨て，**反転点** $\mathbf{p}_{\text{inv}}$ に入れ替えます。このとき今までの**第二最悪点** $\mathbf{p}_m$ が新しい**最悪点** $\mathbf{p}_{m+1}$ になり，**最良点** $\mathbf{p}_1$ は変化しません。$\left\{\mathbf{p}_2, \cdots, \mathbf{p}_{m-1}\right\}$ と $\mathbf{p}_{\text{inv}}$ とで新しく序列化を施します。この序列化では $\left\{\mathbf{p}_2, \cdots, \mathbf{p}_{m-1}\right\}$ と $\mathbf{p}_{\text{inv}}$ とを比較する必要があります。序列化が完了すれば，このステップは終了し（３）に戻ります。

（６）**反転点** $\mathbf{p}_{\text{inv}}$ が今までの**最良点** $\mathbf{p}_1$ より良い場合，つまり，$f(\mathbf{p}_{\text{inv}}) < f(\mathbf{p}_1)$

のとき，**反転拡大** (inversion & expansion) **点** $\mathbf{p}_{\text{expand}}$ を求め，函数値 $f(\mathbf{p}_{\text{expand}})$ を計算します。**反転拡大点** $\mathbf{p}_{\text{expand}}$ は次式で得られます。

$$\mathbf{p}_{\text{expand}} = \mathbf{p}_0 + 2\left(\mathbf{p}_0 - \mathbf{p}_{m+1}\right) = 3\mathbf{p}_0 - 2\mathbf{p}_{m+1}$$

二変数函数の場合の**反転拡大**の操作は， Fig.8.3.3 に示すようになります。

（７）**反転拡大点** $\mathbf{p}_{\text{expand}}$ が**反転点** $\mathbf{p}_{\text{inv}}$ より良い場合，つまり

$$f(\mathbf{p}_{\text{expand}}) < f(\mathbf{p}_{\text{inv}}) < f(\mathbf{p}_1)$$

のとき，**最悪点** $\mathbf{p}_{m+1}$ を捨て，**反転拡大点** $\mathbf{p}_{\text{expand}}$ を採用することとして，序列を更新します。この序列更新は，$\mathbf{p}_{m+1} \leftarrow \mathbf{p}_m$，$\mathbf{p}_m \leftarrow \mathbf{p}_{m-1}$，…，$\mathbf{p}_2 \leftarrow \mathbf{p}_1$，$\mathbf{p}_1 \leftarrow \mathbf{p}_{\text{expand}}$ とするだけなので，比較は必要ありません。序列更新が完了すれば，（３）に戻ります。

（８）**反転拡大点** $\mathbf{p}_{\text{expand}}$ が**反転点** $\mathbf{p}_{\text{inv}}$ より悪い場合，つまり

$$f(\mathbf{p}_{\text{inv}}) \le f(\mathbf{p}_{\text{expand}}) < f(\mathbf{p}_1)$$

のとき，**最悪点** $\mathbf{p}_{m+1}$ を捨て，**反転点** $\mathbf{p}_{\text{inv}}$ を採用し，序列を更新します。この序列更新は，$\mathbf{p}_{m+1} \leftarrow \mathbf{p}_m$，$\mathbf{p}_m \leftarrow \mathbf{p}_{m-1}$，…，$\mathbf{p}_2 \leftarrow \mathbf{p}_1$，$\mathbf{p}_1 \leftarrow \mathbf{p}_{\text{inv}}$ とするだけで，比較は必要ありません。序列更新が完了すれば，（３）に戻ります。

（９）**反転点** $\mathbf{p}_{\text{inv}}$ が**第二最悪点** $\mathbf{p}_m$ より悪い場合，つまり

$$f(\mathbf{p}_m) \le f(\mathbf{p}_{\text{inv}})$$

のとき，**一次元収縮** contraction を施し**収縮点** $\mathbf{p}_{\text{contract}}$ を求め，函数値 $f(\mathbf{p}_{\text{contract}})$ を計算します。**収縮点** $\mathbf{p}_{\text{contract}}$ は以下の式で得られます。

$$\mathbf{p}_{\text{contract}} = \mathbf{p}_0 - 0.5\left(\mathbf{p}_0 - \mathbf{p}_{m+1}\right) = 0.5\mathbf{p}_0 + 0.5\mathbf{p}_{m+1}$$

二変数函数の場合の**収縮**の操作は， Fig.8.3.3 に示すようになります。

（１０）**収縮点** $\mathbf{p}_{\text{contract}}$ が**最悪点** $\mathbf{p}_{m+1}$ より良い場合，つまり

$$f\left(\mathbf{p}_{\text{contract}}\right) < f\left(\mathbf{p}_{m+1}\right)$$

のとき，**最悪点** $\mathbf{p}_{m+1}$ を捨て，**収縮点** $\mathbf{p}_{\text{contract}}$ を採用し， $\left\{\mathbf{p}_1, \cdots, \mathbf{p}_m\right\}$ と $\mathbf{p}_{\text{contract}}$ とで頂点の序列を更新します。この序列更新では $\left\{\mathbf{p}_1, \cdots, \mathbf{p}_m\right\}$ と $\mathbf{p}_{\text{contract}}$ とを比較する必要があります。序列更新が完了すればこのステップは終了し（３）に戻ります。

（１１）**収縮点** $\mathbf{p}_{\text{contract}}$ が**最悪点** $\mathbf{p}_{m+1}$ より悪い場合，つまり

$$f\left(\mathbf{p}_{m+1}\right) \leq f\left(\mathbf{p}_{\text{contract}}\right)$$

のとき，**最良点** $\mathbf{p}_1$ を中心に**縮小** reduction を施します。**縮小** reduction の操作は**反転** inversion，**反転拡大** inversion & expansion，**収縮** contraction と異なり，**最良点** $\mathbf{p}_1$ 以外の全ての点 $\{\mathbf{p}_2, \cdots, \mathbf{p}_{m+1}\}$に施される操作で，以下の式で表されます。

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - 0.5\left(\mathbf{p}_i - \mathbf{p}_1\right) = 1.5\ \mathbf{p}_i - 0.5\ \mathbf{p}_1$$

$$(i = 2,3,\cdots, m+1)$$

函数値 $f(\mathbf{p}_i)$ $(i = 2,3,\cdots, m+1)$ を計算し直し，序列化 (sorting) を施し，（３）に戻ります。

二変数函数の場合の**縮小**の操作を図示すれば， Fig. 8.3.4 のようになります。

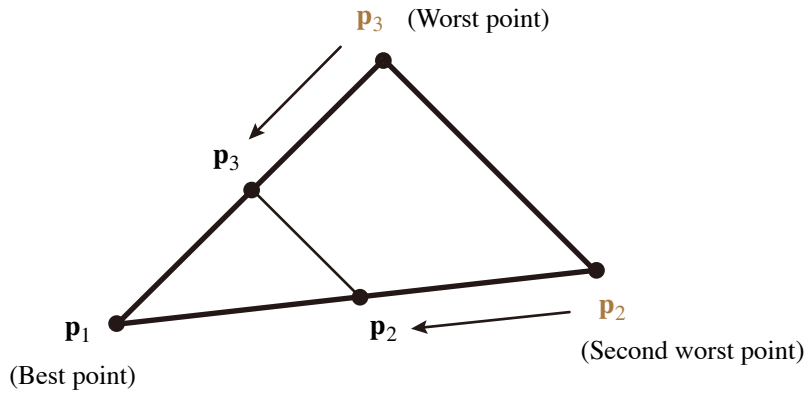

Fig. 8.3.4　ネルダー・ミード法の「縮小 reduction」操作

ネルダー・ミード法で「**極小**」が求まることは証明されているそうですが，シンプレックスの初期配置によって「**偽最小**」に陥る場合があることは，**黄金分割法**と同じです。

補足

（補足 8.A）最適化と函数の最大・最小（←）

「**最適化** optimization とは函数の**最大** maximum あるいは**最小** minimum を求めるのと同じことだ」という言い方には，少し紛らわしさがあるかもしれません。もう少し詳しい表現のしかたをすれば「函数が最大あるいは最小になるときの変数の値を求める」ということで，最大値あるいは最小値を求めるということとは意味が違います。（←）

（補足 8.B）曲線当てはめ curve fitting と回帰 regression（←）

曲線あてはめ curve fitting と**回帰** regression とでは少し意味合いは違うのですが，実際にはほとんど同じ意味で使われます。

「直線をあてはめる」場合でも「曲線当てはめ」という語を使う場合があることに違和感を持つ人もいるかもしれませんが，「数学的には直線は曲線の一種である」とも言えるので，そのことは，あまり気にする必要はないでしょう。

語源からは “regress” は「本来の位置に戻る」という意味合いで，「ノイズや統計的な変動の影響を受けて『本来の値』からずれてしまっている実験データに加工をして『本来の値』に戻してやる」ようなニュアンスがありますが，実際の回帰分析 regression analysis で行われることは「実験データに計算曲線をあてはめる」方に近いと思います。日本での状況はやや特殊で，数学の分野での専門用語の多くに，伝統的に漢語風の表現が用いられてきたので「あてはめ」という和語を用いることに抵抗を持つ人もいるでしょう。“Curve fitting” は中国簡体字では「曲线拟合」と翻訳され，「曲线」は曲線の意味，「拟」は日本で用いられる漢字では「擬」に近く「曲線擬合」と翻訳すれば良いかもしれません。“Regression” は中国簡体字では「回归」と翻訳され，日本語の「回帰」と同じです。（←）

（補足 8.1.A）準ニュートン法 quasi-Newton method（←）

ニュートン法 Newton’s method（ニュートン・ラフソン法 Newton-Raphson method）によって**多変数函数**の方程式を解く場合，「多変数函数の微分」に相当する**勾配** gradient （∇）を使います。**多変数函数**の**最適化**では「多変数函数の２階微分」に相当する**ヘッセ行列** Hessian matrix （**ヘシアン** Hessian）（$\nabla \otimes \nabla = \nabla \nabla^{\mathrm{T}}$）を使います。**準ニュートン法** quasi-Newton method は，勾配やヘシアンの厳密な値を使うわけではないという共通の特徴があります。準ニュートン法の中で過去に比較的よく使われていた方法に，**パウエル法** Powell’s method （パウエルの共役方向法 Powell’s conjugate direction method あるいはダビドン・フレッチャー・パウエル法 Davidon-Fletcher Powell method）がありますが，現在この系統のアルゴリズムとしては，**ブロイデン・フレッチャー・ゴールドファウプ・シャンノ法** Broyden-Fletcher-Goldfarb-Shanno method **（BFGS 法）**の使われる場合が多いようです。

最小自乗法では**レヴェンバーグ・マーカート法** Levenberg-Marquardt **法**の使われる場合が多いでしょう。

ニュートン法系の方法には**最急降下法** gradient descent method (steepest descent method) と**共役勾配法** conjugate gradient method**（CG 法）**と呼ばれる区別もありますが，多変数函数の一次元的な最小化の方向として勾配の最も急な**最急降下方向**を選ぶか，それとは異なる**共役勾配方向**を選ぶかという意味の違いであり，最急降下法と共役勾配法のそれぞれが単独で意味のあるアルゴリズムということではありません。（←）

（補足 8.2.A）黄金長方形（←）

「黄金長方形が最も美しく見える長方形だ」と言う人もいるようですが，実際のところ，見た目では「1 : 1.6 = 5 : 8 の比を持つ長方形」と区別することはできないでしょう。「黄金比」が「美しく見える」として，「1 : 1.62 の方が 1 : 1.60 より美しく見える」と主張することには無理がありそうです。

縦 100 px, 横 162 px の長方形

縦 100 px, 横 160 px の長方形

(←)

（補足 8.2.B）黄金比とフィボナッチ数列（←）

フィボナッチ Fibonacci **数列**は，「新しい項」が「前の項」と「さらにもう一つ前の項」の和で表される 数列で,

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \cdots$$

という数の並び方をします。となりあうフィボナッチ数の比は黄金比に近づくことが知られています。たとえ ば,

$$\frac{0}{1} = 0,\ \frac{1}{1} = 1,\ \frac{1}{2} = 0.5,\ \frac{2}{3} = 0.666\,66\cdots,\ \frac{3}{5} = 0.6,\ \frac{5}{8} = 0.625,\ \frac{8}{13} = 0.615\,38\cdots,$$

$$\frac{13}{21} = 0.619\,04\cdots,$$

などとなります。

第 n 番目のフィボナッチ数を F_n として，隣り合う 2 つの数の比の極限値を

$$x = \lim_{n\to\infty} \frac{F_{n-1}}{F_n} \tag{8.2.B.1}$$

とすれば,

$$x = \lim_{n\to\infty} \frac{F_{n-1}}{F_{n-1}+F_{n-2}} = \lim_{n\to\infty} \frac{1}{1+\dfrac{F_{n-2}}{F_{n-1}}} = \frac{1}{1+x} \tag{8.2.B.2}$$

の関係から

$$x^2 + x - 1 = 0 \tag{8.2.B.3}$$

なので，$0 < x$ のとき,

$$x = \frac{-1+\sqrt{5}}{2} = 0.618\,03\cdots$$

という解を持つことは，自然に導かれます。 (←)

（補足 8.2.C）レオナルド・ダヴィンチのウィトルウィウス的人体図と黄金比（←）

「レオナルド・ダヴィンチの『ウィトルウィウス的人体図』と呼ばれる作品に描かれている円の半径と正方形の辺の長さの比は黄金比に等しい」という説もありますが，実際に黄金比に対応する円を描けば，Fig. 8.2.C.1 の右側の図中に青線で示すように，ダ・ヴィンチの描いた円（赤線）から，かなりずれていることがわかります (http://www.crl.nitech.ac.jp/~ida/education/VitruvianMan/index-j.html)。

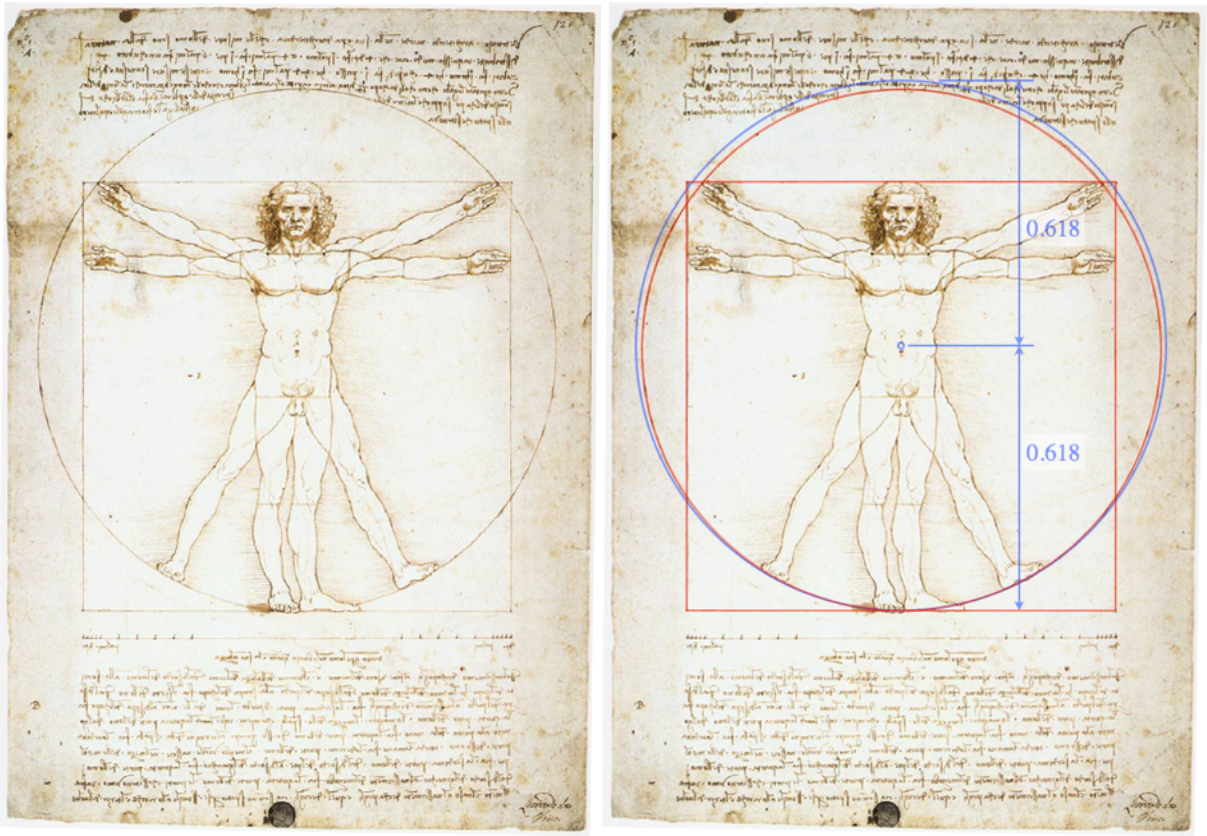

Fig. 8.2.C.1　レオナルド・ダヴィンチのウィトルウィウス的人体図（左）。描かれている円の半径と正方形の辺の長さの比が黄金比に等しいと言う説には無理がある（右）。

（←）

（補足 8.2.D）黄金分割法の別バージョン（←）

以下のようにすれば，はじめの下限 a と上限 d での函数値 $f(a)$ と $f(d)$ のうち，どちらか一回分の函数計算を節約することができます。

(i)　区間の下限を a, 上限を d とする。

(ii)　以下の式に従って，区間を黄金分割する二つの点 b，c をとる。

$$b = \frac{a}{\phi} + \left(1 - \frac{1}{\phi}\right) d$$

$$c = \left(1 - \frac{1}{\phi}\right) a + \frac{d}{\phi}$$

また，$y_2 = f(b)$，$y_3 = f(c)$ の値を計算する。

(iii) もし $y_2 = f(b) < y_3 = f(c)$ なら，最小となるのは $[a, c]$ の区間内だから，$y_1 = f(a)$ の値を計算し，以下の代入：$d \leftarrow c$，$c \leftarrow b$，$y_4 = f(d) \leftarrow y_3 = f(c)$，$y_3 = f(c) \leftarrow y_2 = f(b)$ を行い，$b = a/\phi + (1 - 1/\phi)d$ として $y_2 = f(b)$ の値を計算する。
$y_3 = f(c) \leq y_2 = f(b)$ なら，最小となるのは $[b, d]$ の区間内なので，$y_4 = f(d)$ の値を計算し，以下の代入：$a \leftarrow b$，$b \leftarrow c$，$y_1 = f(a) \leftarrow y_2 = f(b)$，$y_2 = f(b) \leftarrow y_3 = f(c)$ を行い，$c = (1 - 1/\phi)a + d/\phi$ として $y_3 = f(c)$ の値を計算する。

(iv) $y_2 = f(b) < y_3 = f(c)$ なら，最小となるのは $[a, c]$ の区間内だから，以下の代入：$d \leftarrow c$，$c \leftarrow b$，$y_4 = f(d) \leftarrow y_3 = f(c)$，$y_3 = f(c) \leftarrow y_2 = f(b)$ を行い，$b = a/\phi + (1 - 1/\phi)d$ として $y_2 = f(b)$ の値を計算する。$y_3 = f(c) \leq y_2 = f(b)$ なら，最小となるのは $[b, d]$ の区間内なので，$y_4 = f(d)$ の値を計算し，以下の代入：$a \leftarrow b$，$b \leftarrow c$，$y_1 = f(a) \leftarrow y_2 = f(b)$，$y_2 = f(b) \leftarrow y_3 = f(c)$ を行い，
$c = (1 - 1/\phi)a + d/\phi$ として $y_3 = f(c)$ の値を計算する。

(v) (iv) に戻る。

ただし，コードが冗長になるか，余分な条件判断を挿入しなければならなくなるのに対して，節約できる計算量は無視できるほどなので，自分でコードを書く場合に，わざわざこの方法をとる必要はないでしょう。 (←)

（補足 8.2.E）黄金分割法での最適化に失敗する例（←）

黄金分割法で**極小** local minimum が求められることは保証されていますが，**最小** minimum を含む範囲から探索をはじめたとしても，必ず最小が求められるとは限りません。たとえば Fig. 8.2.E.1 に示すような

$$f(x) = -\exp\left[-10\,(x - 0.5)^4\right] - 1.2\exp\left[-10\,(x - 1.5)^2\right] \tag{8.2.E.1}$$

のような函数について，はじめに $x \in [0, 2]$ として黄金分割を開始すると $x = 0.538\,108$ という偽最小 false minimum に陥ってしまいます。

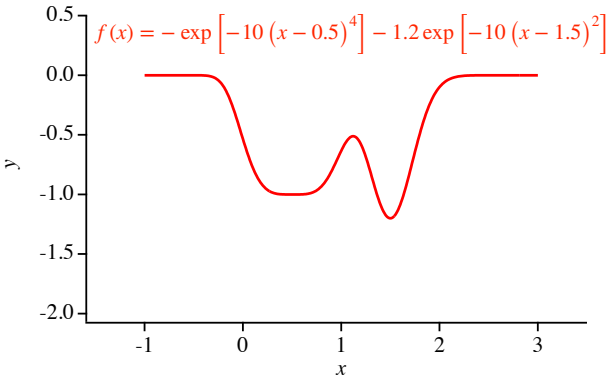

Fig. 8.2.E.1　式 (8.2.E.1) で表される函数の形状。$x \simeq 1.5$ で最小となるが，$x \simeq 0.5$ にも極小を持つ。

式 (8.2.E.1) の函数について，はじめに例えば $x \in [1, 2]$ として最小を含む狭い範囲で黄金分割を開始すれば，正しい最小が求められます。 (←)

（補足 8.2.F）ブレントの方法　Brent's method（←）

ある**標本点**位置 x_0 での**函数値** $y_0 = f(x_0)$とその**微分** $f'(x_0)$，**二階微分** $f''(x_0)$ の値がわかれば，点 (x_0, y_0) を通り，曲線 $y = f(x)$ と傾きも曲率も等しい放物線：

$$y = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(x_0) \tag{8.2.F.1}$$

を一義的に確定することができます。そのような放物線の傾きが 0 となる場所（底の位置）を求めることは「**放物線予測**」と呼ばれます。このことは函数の**微分** $y' = f'(x)$ について「$f'(x) = 0$ の方程式の解を求める」ための**ニュートン・ラフソン 法** Newton-Raphson method を適用するのと同じことであり，収束が始まれば**黄金分割法**より高い効率で最小値を見つけられます。

一方で， Fig. 8.2.2 にグラフを示した式 (8.2.5) のような函数，Fig. 8.2.E.1 にグラフを示した式 (8.2.E.1) のような函数では，二階微分 $f''(x)$ の値が負となる（上に凸の）領域も $f''(x)$ の値が 0 に近い領域も存在します。標本点位置がこれらの領域に位置すれば，極小位置を求めるはずの「放物線予測」が，「極大位置」や「無限に遠い位置」を予測することになるなど，まったく役に立たないことになります。

ブレントの方法 Brent's method は，予測位置での函数値に基づいて，**黄金分割**と**放物線予測**を切り替えながら最小化を行う考え方に基づいており，実際によく使われる方法です。

ブレントの方法も**局所的な最適化手法**のうちの一つであり，この方法で大域的な最適化が可能というわけではありません。（←）