制約付き最適化問題に対する進化型アルゴリズムのためのフィルターに基づく個体の順位付け法

西　孝浩

摘要

制約付き最適化問題は, 工学や経済学など様々な分野に現れる重要な問題である．制約付き最適化問題を効率よく解く手法として，逐次 2 次計画法や内点法がよく用いられている．しかし，それらの手法を適用するためには，目的関数や制約関数が微分可能な関数で定式化される必要がある．また，それらの手法によって大域的最適解を見つけられる保証はない．

目的関数値のみを用いて制約なし最適化問題の大域的最適解を求める手法に，進化型アルゴリズムがある．進化型アルゴリズムとは，多数の探索点 (個体) を生成し，ある順位付けに従ってそれらの個体集合の中から良い個体を選び出し，その個体群を用いて次の探索点を生成するという集合ベースのアルゴリズムである．

本報告書では，進化型アルゴリズムを制約付き最適化問題に適用することを考える．その際重要となるのが，個体の順位付け法である．これまでに提案された個体の順序付け手法は複雑な実行可能集合を持つ問題に対してはあまり有効ではなかった．最近，フィルター法と呼ばれる探索点の評価手法が提案され，点ベースのアルゴリズムにおいて，その有効性が報告されている．そこで，本報告書では進化型アルゴリズムに適したフィルター法に基づく新しい個体の順位付け手法を提案する．さらに，数値実験において，進化型アルゴリズムの 1 つである CMA 進化戦略を用いて制約付き最適化問題を解く．提案する順序付け手法を用いた場合と既存の順序付け手法を用いた場合とで比較，考察を行い，提案した順位付け手法の有効性を議論する．

目次

1 序論

本報告書では次の制約付き最小化問題を考える．

$$\text{(P)} \quad \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) = 0,\ i \in \mathcal{E} \\ & c_i(x) \leq 0,\ i \in \mathcal{I} \end{cases}$$

ここで f, c_i は $\mathbb{R}^n \to \mathbb{R}$ の線形あるいは非線形な関数であり，$\mathcal{E}$,$\mathcal{I}$ はそれぞれ等式制約と不等式制約を表す添字集合である．また，f や c_i の連続性，微分可能性や凸性は仮定しない．目的関数 f と制約関数 c_i が微分可能であれば，内点法や逐次 2 次計画法を用いることで局所的最適解を求めることができる．しかし，それらのアルゴリズムは f や c_i の微分値を必要とするので，関数評価に時間がかかる問題や微分情報を得ることができない問題には適用ができない．本報告書の目的は f, c_i の関数値のみを用いて，問題 (P) の大域的最適解を効率よく求める手法を提案することである．

制約なし最小化問題に対しては，目的関数値のみを用いるアルゴリズムが数多く提案されている．局所探索に基づく手法としては Nelder-Mead 法などの直接探索法がある [12]．直接探索法は実装が容易なため広く用いられている．しかし，求まる解は初期点に依存するため，初期点を適切に選ばないとあまり良くない点に収束してしまうことがある．一方，大域的最適解を求めるための手法としては，進化型アルゴリズムがある [4, 9, 15]．進化型アルゴリズムとは，多数の探索点 (個体) を生成し，ある順位付けに従ってそれらの個体集合の中からある良い個体を選び出し，その個体群を用いて次の探索点を生成するという集合ベースの探索アルゴリズムである．進化型アルゴリズムは厳密な大域的最適解を求めることはできないが，初期点に関係なく良好な近似解を現実的な時間で求めることができる．そして，その性能は探索点の個数，生成法，順位の付け方によって決まる．制約なし最小化問題に対する進化型アルゴリズムの例として，遺伝的アルゴリズム (Genetic Algorithm，以下 GA) [4] や CMA 進化戦略 (Evolution Strategy with Covariance Matrix Adaptation) が知られている．CMA 進化戦略とは，各反復で探索点の集合を多変量正規分布を用いて生成し，その分布の平均値と共分散行列を更新しながら解を探索していく進化型アルゴリズムである [1, 8, 13]．

本来 GA や CMA 進化戦略などの進化型アルゴリズムは制約なし最小化問題に対する解法として考え出されたものであるが，近年，制約付き最小化問題に対して進化型アルゴリズムを拡張する手法がいくつか提案されている．その際に問題となるのが生成された探索点の順位付けである．制約なし最小化問題の場合は，目的関数値の小さい順に探索点の順位付けを行えばよい．しかし，制約付き最小化問題では，目的関数値を小さくすることと同時に，制約条件を満たすことを考えなければならない．したがって，目的関数は小さいものの制約条件を満たしている点と，制約条件は満たしているものの目的関数がそこまで小さくない点のどちらの方が“良い” 探索点であるかを判断する必要がある．

このような問題を解決する最も簡単な手法はペナルティ関数を用いることである．ペナルティ関数は $p(x) = f(x) + \rho h(x)$ と定義される関数である．ただし，$h(x)$ は制約違反関数と呼ばれる点 x の実行不可能性を測る関数であり，一般に，$h(x) = \sum_{i \in \mathcal{E}} |c_i(x)| + \sum_{i \in \mathcal{I}} \max\{0, c_i(x)\}$ で定義される．ρ は制約を違反することに対するペナルティの大きさを調整するパラメータであり，ペナルティパラメータと呼ばれる．ペナルティ関数を用いた手法ではペナルティパラメータの設定が難しく，ペナルティパラメータを適切に選ばないと実行可能解すら求まらない．その欠点を克服する手法として，Deb [2] は次のように探索点に順位付けを行う手法を提案した．それは，まず制約条件を満たす探索点の中で目的関数 f の値の低いものから順位をつけ，次に，残っ

た探索点 (制約条件を満たさない点) の中で制約違反関数 h の値の低いものから順位をつけるというものである．この手法は実行可能点が求めやすい問題に対しては特に有効であり，現在最もよく使用されている順位付け法である．しかし，実行可能点を求めることが困難な問題に対しては，制約違反関数 h の値の低い探索点から順番に順位をつけるだけになってしまい，目的関数 f の値を改善する探索点が生成されないことがある．

上記の問題を解決するため，本報告書ではフィルター法 [5, 6] のアイデアを用いた順位付け法を提案する．フィルター法は制約付き最小化問題を f と h の 2 つの関数を最小化する多目的最小化問題に置き換え，パレート最適性と呼ばれる概念を用いて探索点の良さを判断する．多目的最適化問題に対する進化型アルゴリズムについては既にパレート最適性に基づくいくつかの順位付け法が提案されている．しかし，制約付き最適化問題では制約条件を最終的に満たさなければならないため，それらの多目的最適化の方法をそのまま適用することはできない．最近 Hedar と Fukushima [11] は焼きなまし法 (Simulated anealing, 以下 SA) において近傍内の点の順位付けにフィルター法のアイデアを用いた手法を提案した．しかし，この順位付け法は点ベースの手法である SA のために開発されたものであり，集合ベースの進化型アルゴリズムにそのまま用いるには適さない．そこで本報告書では集合ベースの進化型アルゴリズムに適したフィルター法に基づく順位付け法を提案する．さらに，いくつかの数値実験を行い，その結果から提案手法の有効性を議論する．

本報告書の構成は以下のとおりである．まず，2.1 節で制約なし最小化問題に対する進化型アルゴリズムについて説明し，2.2 節で進化型アルゴリズムの制約付き最適化問題への拡張法を述べる．次に，3 節で既存の探索点の順位付け法とその問題点について述べる．4 節でフィルターに基づいた新しい探索点の順位付け法を提案する．5 節で提案手法を CMA 進化戦略と組み合わせた手法の数値実験結果を報告し，その考察を与える．最後に，6 節において結論を述べる．

2 進化型アルゴリズム

本節では制約なし最適化問題，制約付き最適化問題，それぞれに対する進化型アルゴリズムの基礎的な概念を紹介する．

2.1 制約なし最適化問題に対する進化型アルゴリズム

本節では制約なし最小化問題 (Q) に対する進化型アルゴリズムについて述べる．

$$\text{(Q)} \quad \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathbb{R}^n \end{cases}$$

制約なし最小化問題に対する進化型アルゴリズムでは，各反復 (世代と言う) において λ 個の探索点を生成し，目的関数 f の値の小さい点から順位付けを行う．次に，その順位付けに従って μ 個の探索点を選び出す．選択された μ 個の探索点の情報を用いて次の探索点を生成する．アルゴリズムの概要は次のとおりである．

Step 0： 世代 $g = 0$ とし，初期集団 $P(0)$ を設定する．
Step 1： 集団 $P(g)$ を用いて，**生成規則**に従い λ 個の探索点を生成する．
Step 2： **順位付け規則**に従って探索点に順位をつける．
Step 3： Step 2 で求めた順位と，**選択規則**に従って μ 個の点を選択し，集合 $P(g+1)$ とする．
Step 4： 終了条件を満たせば終了．そうでなければ $g = g + 1$ とし Step 1 に戻る．

探索点の生成規則，探索点の順位付け規則，点の選択規則を変えると，異なる進化型アルゴリズムが構成できる．進化型アルゴリズムの一つである CMA 進化戦略 (付録 A 参照) では，各反復において集団 $P(g)$ 内の点の重み付き平均や，重み付き共分散行列を用いた n 次元の多変量正規分布を用いて次の世代の探索点を生成する．さらに，各探索点は目的関数 f の値の小さいものから順に順位付けされ，順位の高い方から順に μ 個の点が選択される．

2.2 制約つき最適化問題に対する進化型アルゴリズム

本節では，進化型アルゴリズムを制約つき最適化問題 (P) に適用することを考える．

問題 (P) は次のように簡単に表すことができる．

$$(\mathrm{P}^*) \quad \begin{cases} \text{minimize} & f(x) \\ \text{subjext to} & h(x) = 0 \end{cases}$$

ただし，h は次のように定義される関数である．

$$h(x) = \sum_{i \in \mathcal{E}} |c_i(x)| + \sum_{i \in \mathcal{I}} \max\{0, c_i(x)\}$$

以下では，h を制約違反関数と呼ぶことにする．制約付き最適化問題 (P) に対する進化型アルゴリズムの概要は次のとおりである．

Step 0： 世代 $g = 0$ とし，初期集団 $P(0)$ を設定する．
Step 1： 生成規則に従い，集団 $P(g)$ を用いて λ 個の探索点を生成する．
Step 2： **探索点に目的関数 f の値，制約違反関数 h の値を用いて順位付けを行う．**
Step 3： λ 個の探索点のうち，順位付けを参考に, 選択規則に従って μ 個の点を選択する．
Step 4： 選択した μ 個の点を集合 $P(g+1)$ とする．
Step 5： 終了条件を満たせば終了．そうでなければ $g = g + 1$ とし Step 1 に戻る．

2.1 節で述べたように，探索点の生成法や，探索点の順位の付け方，探索点の生成の仕方を変えると，(P) に対する異なる進化型アルゴリズムを構成できる．

問題 (P) においては目的関数 f の最小化と制約違反関数 $h(x) = 0$ を満たすことの両方を考えなければならないため，目的関数は小さいものの制約条件を満たしている点と，制約条件は満たしているものの目的関数がそこまで小さくない点のどちらの方が “良い” 探索点であるかを判断する必要がある．よって，制約なし最小化問題の場合とは異なり，探索点の順位付けには様々な基準が考えられる．そして，順位付けの良し悪しが進化型アルゴリズムの性能を大きく左右する．次節では，既存の探索点の順位付け手法を紹介し，その問題点を指摘する．

3 既存の個体の順位付け手法

本節では問題 (P) に対する進化型アルゴリズムにおける既存の 2 つの順位付け手法を紹介する．1 つは古くからあるペナルティ関数を用いた手法であり，もう一つは現在最もよく使用されている Deb の手法 [2] である．

3.1 ペナルティ関数を用いた手法

ペナルティ関数を用いた手法では，まず制約付き最小化問題 (P) を次の制約なし最小化問題 (P') に置き換える．

$$\text{(P')} \quad \begin{cases} \text{minimize} & p(x) = f(x) + \rho h(x) \\ \text{subject to} & x \in \mathbb{R}^n \end{cases}$$

ここで $\rho(> 0)$ はペナルティパラメータと呼ばれ，制約を違反することに対するペナルティの大きさを調整するパラメータである．問題 (P') は制約なし最小化問題であるので，探索点にはは関数 $p(x)$ の値の小さい方から高い順位をつければよい．

この手法はシンプルであり実装もやさしい．しかし，ペナルティパラメータの設定は難しく，ペナルティパラメータを適切に選ばないともとの問題 (P) の実行可能解が求まらないという欠点がある．

3.2 Deb の手法

ペナルティ関数を用いた手法の欠点を克服するため，Deb [2] は次のような順位付けの手法を提案した．

Step1 制約条件を満たす点 ($h(x) = 0$ を満たす点) の中で目的関数 f の値の小さいものから順位をつける．

Step2 残った点 ($h(x) > 0$ となる点) の中で制約違反関数 h の値の小さいものから順位をつける．

この手法では目的関数 f と制約違反関数 h を同時に用いていない．そのためペナルティパラメータを設定する必要がない．この手法は問題 (P) の実行可能解を求めやすいときには比較的良い順位付けを与える．しかし，問題 (P) が等式制約をもつときは，特別な工夫を与えない限り，たいていの探索点では $h(x) > 0$ となる．そのため，この順位付け手法では制約違反関数 h の値の低いものから順位をつけるだけになってしまい，目的関数 f の値を改善する探索点が生成されないという欠点がある．

4　フィルターに基づく個体の順位付け手法

前節での個体の順位付け法の問題点を解決するためにフィルターに基づく個体の順位付け法を提案する．

4.1　フィルター法

まず，制約付き最適化問題 (P) には，2 つの競合する目的があることに注目する．1 つは目的関数 f を最小にすることであり，もう 1 つは制約条件を満たすこと，すなわち制約違反関数 $h(\geq 0)$ が $h(x) = 0$ を満たすことである．このことは次のように多目的最適化問題で表すことができる．

$$\text{(P")} \quad \begin{cases} \text{minimize} & f(x) \\ \text{minimize} & h(x) \end{cases}$$

問題 (P") と一般の多目的最適化問題では，大きく異なる点が 1 つある．それは，問題 (P) の最適解は制約条件をすべて満たされなければならないので，問題 (P") の解を求める上で $h(x) = 0$ となるような点を見つけることを優先すべきことである．

ここで多目的最適化問題における「優越関係」と呼ばれる概念とフィルターと呼ばれる概念 [5, 6] を紹介する．以下では，点 $x^{(k)} \in \mathbb{R}^n$ のかわりに，その関数値である $(f^{(k)}, h^{(k)}) = (f(x^{(k)}), h(x^{(k)}))$ を考える．

定義 4.1 (優越関係) $(f^{(k)}, h^{(k)})$ が $(f^{(l)}, h^{(l)})$ を優越するとは，$f^{(k)} \leq f^{(l)}$ かつ $h^{(k)} \leq h^{(l)}$ が成り立つことをいう．

定義 4.2 (フィルター) フィルターとは，点列集合 $\{(f^{(k)}, h^{(k)})\}$ で，その集合の中から任意に 2 つの要素を取り出してきても優越関係になっていないもののことをいう．

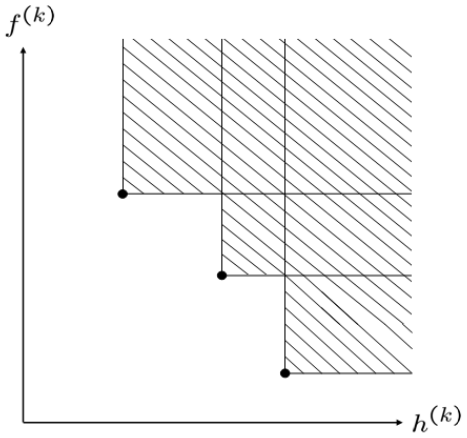


図 1　フィルター

図 1 はフィルターを図示したものである．黒丸がフィルターを構成する点を表し，斜線部分はフィルター内の点によって優越されている部分を表す．

フィルター法とは，新しい探索点 x_{new} が与えられたとき，その探索点の良さをフィルターを用いて判断する手法である．一般に x_{new} がフィルターを構成する点から優越されていなければ，x_{new} はよい探索点と考えることができる．これまでに，フィルター法と逐次 2 次計画法が組み合わされた手法が提案されており，ペナルティ法と逐次 2 次計画法を組み合わせた手法よりも有効であることが報告されている [5, 6]．

4.2 新しい個体の順位付け手法

提案する手法の目的は次のとおりである．

- 目的関数 f と制約違反関数 h をできるだけ “同時に” 最小化する．
- 制約違反関数 h の最小化を目的関数 f の最小化より優先する．

3.2 節で述べたとおり，Deb の手法においては制約違反関数 h の値が 0 でない場合，目的関数 f の値が無視されていた．しかし，本報告書で提案する手法は，フィルター法のアイデアに基づき目的関数 f の最小化と制約違反関数 h の最小化を “同時” に行うことを目指す．そのことによって制約条件が複雑な問題であってもより速く，より良い点に収束することが期待される．この 2 つの目的を達成するために，進化型アルゴリズムにおける探索点の新しい順位付け法を 2 つ提案する．その概要を以下で説明する．

4.2.1 提案手法 1

最初に提案する探索点の順位付け法は次のとおりである．

Step 1 (**フィルターの生成**)
　λ 個の探索点からフィルターを生成する．
Step 2 (**フィルターを構成する点に対する順位付け**)
　フィルターを構成する探索点に対して制約違反関数 h の値の小さいものから順位をつける．
Step 3 (**残った点によるフィルターの生成**)
　順位付けの終えた点を取り除き，残った点で再びフィルターを生成し，Step2 に戻る．

この順位付け法のアイデアを説明する．まず Step1 では，各反復において生成された探索点をもとにフィルターを生成する．フィルターの定義より，各反復においてフィルターを構成する点は，他の探索点によって優越されない．よって，フィルターを構成する点は，各反復における探索点の中で，目的関数 f の値を減少させるという目的と，制約違反関数 h の値を減少させるという目的の 2 つの目的に対して最も “良い” 点群であるといえる．(この関係にある点列を問題 (P”) のパレート最適解という．) よって，これらの探索点には高い順位を与えたい．また，この節の初めに述べたように，制約違反関数 h の最小化を目的関数 f の最小化より優先したい．そこで，Step2 ではフィルターを構成する点列に対して h の値の小さいものから順位をつけている．Step2 で順位付けを終えた点を取り除いたとき，順位付けを終えていない点が残っていれば，それらの点に対して同様の順位付けが行う．これをすべての点に順位が付くまで繰り返す．

この手法は一般の多目的最適化問題において Goldberg が提案した順位付け手法に似ている [7]．この提案手法を本報告書では FPO 法 (Filter Peeling Ordering) と呼ぶことにする．

4.2.2 **提案手法** 2

次に，以下のような探索点の順位付け法を提案する．

Step 1 (**フィルターの生成**)
　λ 個の探索点からフィルターを生成する．
Step 2 (**フィルターを構成する点に対する順位付け**)
　フィルターを構成する探索点に対して制約違反関数 h の値の小さいものから順位をつける．
Step 3 (**フィルターに含まれない探索点に対する順位付け**)
　まだ順位のついていない探索点に関しては「その点を優越している点についている順位の中で最大の順位」+1 の順位をつける．

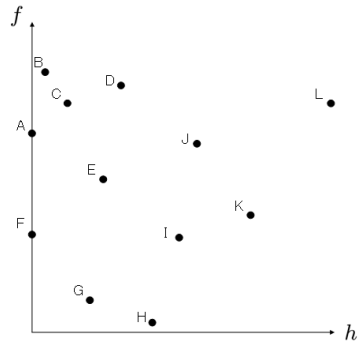
この順位付け法のアイデアを説明する．まず Step1 では，FPO 法と同様に，各反復において生成された探索点をもとにフィルターを生成し，フィルターを構成する点に対して，h の値の小さいものから順位をつける．

次に，フィルターに含まれない点を考える．フィルターに含まれない点は少なくとも 1 つの点に優越されている．優越されている点は目的関数 f の値の減少という面でも，制約違反関数の減少という面でもそれを優越している点よりも劣っている．したがって，優越されている点はそれを優越する点より低い順位を与えられなければならない．そこで Step3 では，「(その点を優越している点についている順位の中で最大の順位)+1」という順位をその点に与える．これによってすべての点に目的を満たす順位を与えることができる．

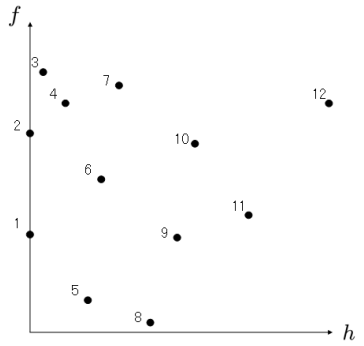
この手法は，特に優越関係をより顕著に表す順位付け法であり，優越関係にない点同士に対しては同じ順位が付くこともある．そこで，この順位付け法を実際に適用する際には，何らかの方法で同じ順位の物の優劣をつける必要がある．この提案手法を本報告書では DRO 法 (Dominance Ranking Ordering) と呼ぶことにする．

FPO 法と DRO 法のどちらの提案手法も，Deb の手法と同じように目的関数 f の値と制約違反関数 h の値を同時に使用しないため，ペナルティパラメータを必要としない．

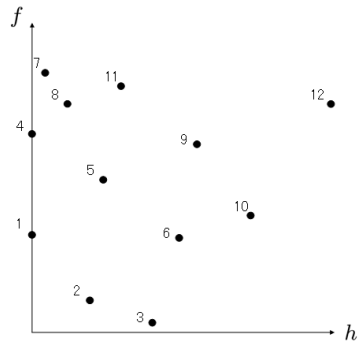
例として，図 2 に生成した探索点とそれに対する Deb の手法，FPO 法，DRO 法による順位付けを示す．図 2 において点 G を点 B や点 C と比べると制約違反関数の値の差はほとんどなく，目的関数の値では大幅に点 G の方が点 B や点 C より優っている．Deb の手法では制約違反関数の値しか見ないため点 B や点 C の方が点 G に比べ高い順位がついているが，FPO 法や DRO 法では点 G の方が点 B や点 C に比べ高い順位がついている．同様に，点 D と点 H を見比べた場合も手法によって順位が異なっている．したがって，提案手法は制約を満たしていない点においても制約違反関数と目的関数の両方を最小化できるような点を選び出しているといえる．そのため，実行可能解を求めることが難しい問題に対しては，Deb の手法に比べ，FPO 法や DRO 法はより速く，よりよい最適解に収束する可能性が増大すると考えられる．



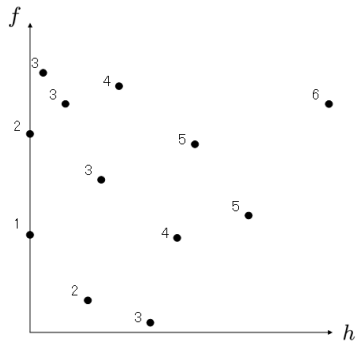
(a) 探索点例



(b) Deb の手法による探索点の順位



(c) FPO 法による探索点の順位



(d) DRO 法による探索点の順位

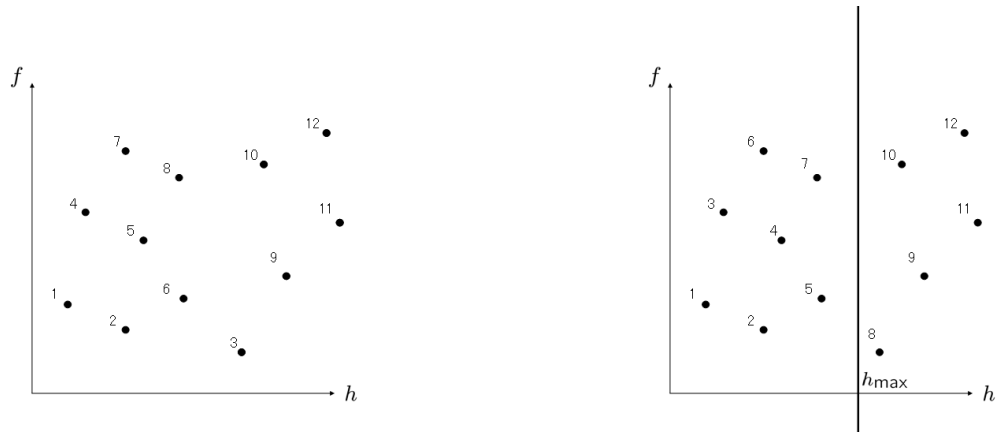
図 2　Deb の手法と提案手法による探索点の順位

4.2.3　実装上の工夫

4.2.1,4.2.2 節で述べた順位付け手法を計算機上で実装する上での工夫を述べる．

制約違反関数値が大きい探索点でも,，目的関数の値が小さいとフィルターに含まれてしまう場合がある．特に FPO 法では，フィルターに含まれている点を優先するため，そのような点があるといつまでたっても収束しない恐れがある．そのようなことを避けるために，制約違反関数がある値 $h_{\max}$ を超える値をとる探索点は，制約違反関数が $h_{\max}$ 以内の値を持つ探索点の順位を付けた後に順位を付けることを考える．

FPO 法を用いて順位付けを行った場合において，この工夫を行わない場合とこの工夫を導入した際の順序を図 3 に示す．図 3 (a) で 3 番目に順位付けされた点が図 3 (b) ではその点の制約違反関数 h の値が $h_{\max}$ を超えているために 8 番に順位付けされているのがわかる．



(a) 工夫なしの順位　　(b) 工夫ありの順位

図 3　$h_{\max}$ を用いた工夫と FPO 法による探索点の順位

5　数値実験

本節では，4 節で提案した順位付け法を用いた数値実験結果を報告する．実験は CPU が 3.4GHz の Pentium4，メモリが 3.6GB の計算機上で行い，アルゴリズムは MATLAB 7 を用いて実装した．

この実験の目的は，制約付き最小化問題における進化型アルゴリズムにおいて，3.2 節で述べた Deb の手法と，提案した 2 つの順位付け手法を比較し，その有効性を検証することである．進化型アルゴリズムの有効性を調べるために次の 2 点について議論する．

- 大域的最適解に収束する割合
- 関数評価回数

関数評価回数は少ない方がよく，大域的最適解への収束する割合は高いほど良い．

本実験では進化型アルゴリズムとして CMA 進化戦略 [8] を用いる．CMA 進化戦略のアルゴリズムの概要は付録 A にまとめてある．

ここで計算機上で CMA 進化戦略のアルゴリズムを実装する際の工夫を述べる．CMA 進化戦略では，各反復において探索点 x を計算機上で乱数によって生成するため，元の問題 (P) の等式制約 $c_i(x) = 0,\ i \in \mathcal{E}$ を厳密に満たす探索点は生成されない．このため，実行可能集合の境界付近で順位付けが不安定になることがある．そこで等式制約 $c_i(x) = 0,\ i \in \mathcal{E}$ を不等式制約 $c_i(x) - \epsilon \leq 0,\ i \in \mathcal{E}$ に置き換える．そのことによって，CMA 進化戦略は計算機上でよりよい性能を発揮する．実験では $\epsilon = 10^{-3}$ とした．また，4.2.3 節で述べた制約違反関数 h の上限 $h_{\max}$ を 100 とした．

CMA 進化戦略の終了条件は，制約違反関数 $h(x) < \epsilon_{\text{stop}}$ でかつ，$\|m^{(g+1)} - m^{(g)}\| < \epsilon_{\text{stop}}$ とした．ただし，$\epsilon_{\text{stop}} = 10^{-8}$ とし，$m^{(g)}$ は CMA 進化戦略の各反復における x の重み付き平均である (付録 A)．すなわち，制約条件を満たし，各反復における探索点集合の平均値の移動が一定値以下になれば終了する．また関数評価回数が 50,000 回を超えた場合は，その時点で打ち切った．打ち切った場合は最適解が求められなかったと判断した．

実験に用いたてスト問題は CUTEr [14] のテスト問題集と論文 [10] のテスト問題から選んだ．表 1 にテス

ト問題の問題名とその問題の変数および制約の数を示す．ただし，各問題の変数の数を n とし，制約の数を m とする．また，制約のうち等式制約の数を m_E とし，不等式制約の数を m_I とする．$m = m_E + m_I$ である．

初期点 $m^{(0)}$ は CUTEr のテスト問題については与えられた初期点を用い，それ以外の場合は一様乱数 (0, 10) に従う n 変量ベクトルを用いる．各問題の大域的最適値 f^* は既知である．そこで各手法で得られた解の目的関数値 f' に対して，$\left|\frac{f'-f^*}{\max\{1,f^*\}}\right| < 0.02$ が成立すれば大域的最適解に収束したとみなした．

表 1　実験に用いた問題の変数の数 n と制約の数 m

問題名	n	m	$(m_E,$	$m_I)$
PrG1	13	35	(0 ,	35)
PrG2	20	42	(0 ,	42)
PrG3	20	41	(1 ,	40)
PrG4	5	16	(0 ,	16)
PrG5	4	13	(3 ,	10)
PrG6	2	6	(0 ,	6)
PrG7	10	28	(0 ,	28)
PrG8	2	6	(0 ,	6)
PrG9	7	18	(0 ,	18)
PrG10	8	22	(0 ,	22)
PrG11	2	5	(1 ,	4)
PrG12	3	729	(0 ,	729)
PrG13	5	13	(3 ,	10)
PrW	3	4	(0 ,	4)
PrP	3	10	(0 ,	10)
PrT	4	14	(0 ,	14)
BOX2	3	1	(1 ,	0)
BT1	2	1	(1 ,	0)
BT2	3	1	(1 ,	0)
BT3	5	3	(3 ,	0)
BT4	3	2	(2 ,	0)
BT5	3	2	(2 ,	0)
BT6	5	2	(2 ,	0)
BT7	5	3	(3 ,	0)
BT8	5	2	(2 ,	0)
BT9	4	2	(2 ,	0)
BT10	2	2	(2 ,	0)
BT11	5	3	(3 ,	0)
BT12	5	3	(3 ,	0)
BT13	5	2	(1 ,	1)
CB2	3	3	(0 ,	3)

問題名	n	m	$(m_E,$	$m_I)$
CB3	3	3	(0 ,	3)
EXTRASIM	2	2	(1 ,	1)
FCCU	19	27	(8 ,	19)
HART6	6	6	(0 ,	6)
HATFLDA	4	4	(0 ,	4)
HS1	2	1	(0 ,	1)
HS24	2	5	(0 ,	5)
HS29	3	1	(0 ,	1)
HS99	7	16	(2 ,	14)
HS100	7	4	(0 ,	4)
HS104	8	21	(0 ,	21)
HS105	8	17	(1 ,	16)
HS106	8	22	(0 ,	22)
HS107	9	14	(6 ,	8)
HS108	9	14	(0 ,	14)
HS109	9	26	(6 ,	20)
HS110	10	20	(0 ,	20)
HS111	10	23	(3 ,	20)
HS112	10	13	(3 ,	10)
HS113	10	8	(0 ,	8)
HS114	10	31	(3 ,	28)
HS116	13	40	(0 ,	40)
HS117	15	20	(0 ,	20)
HS118	15	47	(0 ,	47)
HS119	16	40	(8 ,	32)
LOGROS	2	2	(0 ,	2)
LOTSCHD	12	7	(7 ,	0)
MADSEN	3	6	(0 ,	6)
SUPERSIM	2	3	(2 ,	1)
TAME	2	3	(1 ,	2)
TRY-B	2	3	(1 ,	2)

5.1 実験結果

それぞれの順位付け手法を組み込んだ CMA 進化戦略を用いて各問題を 30 回ずつ解いた．各問題において，それぞれの手法を用いた場合の関数評価回数，大域的最適解への収束回数を表 2 と表 3 に示す．ただし，関数評価回数は大域的最適解が得られたときの平均関数評価回数であり，30 回のうち一度も大域的最適解に収束しなかった手法に対しては関数評価回数を F と表した．

5.2 考察

まず大域的最適解への収束ついて考える．表 2，表 3 より，62 のテスト問題を解き DRO 法が一度も大域的最適解を求めることができなかった問題は 4 つである．一方，FPO 法は 9 つ，Deb の手法では 7 つの問題で一度も大域的最適解を求めることができなかった．また，15 回以上大域的最適解に収束した問題の数は，FPO 法が 42 個，DRO 法が 44 個，Deb の手法が 44 個である．さらに，20 回以上大域的最適解に収束した問題の数は FPO 法が 40 個，DRO 法が 41 個，Deb の手法が 39 個となっている．以上より，DRO 法は大域的最適解が求まる可能性の高いことがわかる．

以下では，最近アルゴリズムの性能を判断するのによく使用される性能指標 [3] を用いて関数評価回数に関する考察を行う．

性能指標の計算方法は次のとおりである．各手法を表す添字を $s \in \mathcal{S}$ として，各問題を表す添字を $p \in \mathcal{P}$ とする．ただし，$\mathcal{S} = \{\text{FPO,DRO,Deb}\}$ であり，$\mathcal{P}$ は解いたテスト問題の集合である．各問題 $p \in \mathcal{P}$ に対して，手法 s により問題 p の大域的最適解を求めるのに要した平均関数評価回数を $F_{p,s}$ とし (手法 s により問題 p が解けない場合 $F_{p,s} = \infty$ とする)，問題 p に対する手法 s の性能比 $r_{p,s}$ を次のように定める．

$$r_{p,s} = \frac{F_{p,s}}{\min\{F_{p,s} : s \in \mathcal{S}\}}$$

すなわち，問題 p^* において最も関数評価回数の少ない手法 s^* に対しては $r_{p^*,s^*} = 1$ となり，それ以外の手法 s に対しては $r_{p^*,s} > 1$ となる．

性能比 $r_{p,s}$ を用いて手法 s の性能度 $\rho_s(\tau)$ を次のように定義する．

$$\rho_s(\tau) = \frac{r_{p,s} \leq \tau\text{を満たす } \mathcal{P} \text{ の要素数}}{\mathcal{P} \text{ の要素数}} \quad (\tau \geq 1)$$

ただし，性能度 ρ_s は $[1, \infty) \to [0, 1]$ の非減少関数である．$\rho_s(1)$ は，手法 s が最も関数評価回数が少なくなる問題の数を，解いたテスト問題の総数で割った値である．すなわち，問題集合 $\mathcal{P}$ の中で手法 s が最も関数評価回数が少なくなる問題の割合となる．また，十分大きな M に対して

$$\rho_s^* = \rho_s(M)$$

とすると，ρ^* は手法 s において大域的最適解を求められる割合となる．

この性能度 ρ_s を用いて順位付け手法を比較する．それぞれの手法に対し，性能度 $\rho_s(\tau)$ を図 4 に示す．図 4 (a) は $1 \leq \tau \leq 2$ の範囲での各手法の性能度 $\rho_s(\tau)$ を，同様に，図 4 (b) は $1 \leq \tau \leq 10$ の範囲での各手法の性能度 $\rho_s(\tau)$ を与えている．

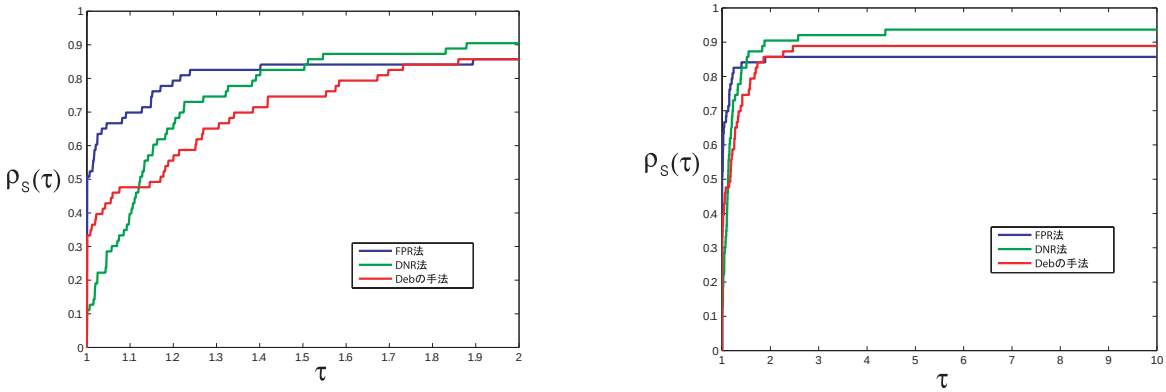
図 4 (a) の $\tau = 1$ 付近の値を比べると，FPO 法の値が最も高くなっている．これは FPO 法の関数評価回数が少ない傾向にあることを示している．実際，$\tau = 1$ では $\rho_{p,\text{FPO}}(1) = 0.4921$， $\rho_{p,\text{DRO}}(1) = 0.1111$，

表 2　実験結果 (関数評価回数，最適解への収束回数)

	FPO 法		DRO 法		Deb の手法	
問題名	関数評価回数	収束回数	関数評価回数	収束回数	関数評価回数	収束回数
PrG1	68271.1	13	90215.1	16	72072.0	13
PrG2	6000.0	3	3168.0	1	4212	1
PrG3	24093.2	30	23293.6	30	27271.6	30
PrG4	F	0	8765.1	30	8398.9	30
PrG5	F	0	9732.0	2	3781.3	3
PrG6	1930.2	30	2074.8	30	2419.0	30
PrG7	22716.7	30	21898.3	30	19785.7	30
PrG8	424.3	25	417.6	25	426.7	25
PrG9	9899.7	30	9414.3	30	8621.1	30
PrG10	42575.0	8	43633.8	8	43060.0	15
PrG11	1583.2	30	1773.0	30	2066.4	30
PrG12	1299.2	5	1578.5	4	2046.3	3
PrG13	F	0	15720.8	10	F	0
PrW	F	0	F	0	F	0
PrP	14281.8	9	14578.0	4	13941.6	10
PrT	2620.3	27	1907.1	27	1870.1	26
BOX2	2851.3	27	3996.6	19	4938.0	7
BT1	810.0	1	972.0	1	1014.0	1
BT2	6154.2	18	6947.5	8	6520.5	4
BT3	5186.4	30	6030.7	30	6582.4	30
BT4	9262.8	19	10403.4	5	11011.0	1
BT5	6279.8	26	7692.7	23	7422.2	16
BT6	5297.7	28	7325.9	16	11995.8	19
BT7	10147.1	23	10592.0	21	10015.5	16
BT8	3321.6	30	3765.1	30	4452.8	30
BT9	4612.3	30	6971.7	30	11377.0	25
BT10	1056.2	29	1040.7	22	1119.0	18
BT11	6208.0	30	7332.0	30	11545.1	30
BT12	14663.1	9	17378.7	3	F	0
BT13	9218.8	29	13856.3	24	15650.3	24

表 3　実験結果 (関数評価回数，最適解への収束回数)

	FPO 法		DRO 法		Deb の手法	
問題名	関数評価回数	収束回数	関数評価回数	収束回数	関数評価回数	収束回数
CB2	2620.1	30	2992.0	30	2730.5	30
CB3	1721.5	30	1986.4	30	2026.7	30
EXTRASIM	1043.6	30	1170.2	30	1266.6	30
FCCU	60762.0	30	74442.4	30	86212.8	30
HART6	1150.5	30	1248.3	30	1172.7	30
HATFLDA	888.8	30	905.1	30	888.5	30
HS1	984.0	30	914.0	30	854.2	30
HS24	959.0	30	1086.0	30	1214.6	30
HS29	2347.6	30	2436.9	30	2331.9	30
HS99	F	0	F	0	F	0
HS100	9071.7	30	9266.4	30	8319.6	30
HS104	5341.3	30	5017.3	30	4939.7	30
HS105	23011.0	30	21074.3	30	19197.0	30
HS106	31968.0	5	44498.3	6	45355.0	12
HS107	40564.0	30	40143.7	30	33339.7	30
HS108	F	0	F	0	F	0
HS109	1427.8	30	1645.6	30	1636.0	30
HS110	F	0	F	0	F	0
HS111	1459.0	30	1493.3	30	1429.0	30
HS112	87025.0	6	93220.0	3	70266.7	3
HS113	15648.7	30	16903.0	30	15383.0	30
HS114	22024.0	30	20884.7	30	18812.0	30
HS116	61908.6	22	54917.5	8	F	0
HS117	F	0	89276.0	1	20383.0	1
HS118	F	0	165854.0	13	161827.0	21
HS119	94104.0	1	145500.0	29	130293.0	27
LOGROS	73902.9	14	135257.0	30	114840.0	30
LOTSCHD	64048.1	9	120294.0	28	101427.0	28
MADSEN	2922.7	30	2852.0	30	2877.0	30
SUPERSIM	931.2	30	948.6	30	965.2	30
TAME	487.4	30	537.8	30	585.0	30
TRY-B	1270.8	29	1612.6	30	2125.0	26



(a) 性能度 $\rho_s(\tau)$ $(1 \leq \tau \leq 2)$　　(b) 性能度 $\rho_s(\tau)$ $(1 \leq \tau \leq 10)$

図 4　性能度による各手法の比較

$\rho_{p,\mathrm{Deb}}(1) = 0.3333$ である．すなわち，問題集合 $\mathcal{P}$ において 49.21% の問題において FPO 法が最も関数評価回数の少ない手法であり，11.11% の問題においては DRO 法が，33.33% の問題においては Deb の手法が最も関数評価回数が少ない手法であることを示している．また，$\tau = 1.25$ を考えると，性能指標 ρ は最小関数評価回数の 1.25 倍より少ない関数評価回数で解ける問題の割合を表し，$\rho_{p,\mathrm{FPO}}(1.25) = 0.8254$, $\rho_{p,\mathrm{DRO}}(1.25) = 0.7302$，$\rho_{p,\mathrm{Deb}}(1.25) = 0.5873$ である．すなわち，最も少ない関数評価回数から 1.25 倍以内の関数評価回数で収束する問題の割合が，FPO 法では 82,54% であり，DRO 法では 73.02% であり，Deb の手法では 58.73% である．以上のより，FPO 法は他の手法に比べ関数評価回数が少ない傾向にあるということが伺える．

6　結論と今後の課題

本報告書では，フィルター法のアイデアを用いて制約付き最適化に対する進化アルゴリズムにおける順位付け手法を 2 つ提案した．数値実験において，FPO 法が既存の手法に比べて関数評価回数を減少させる傾向にあること，また，DRO 法を用いると大域的最適解が求まる可能性が高まることがわかった．

今後の課題としては次のようなことが挙げられる．

- 各手法が有効となる問題の特徴を調べる．
- 各手法において，目的関数 f の値と制約違反関数 h の値の減少の速度とその関係を調べる．
- GA や PSO など今回用いた CMA 進化戦略以外の進化型アルゴリズムに対しても有効な順序付けであるかどうかを確かめる．
- 大域的最適解に収束しない場合のアルゴリズムの挙動はどうなるのか (収束はするのか，収束するとしたらどの程度良い点に収束するのか) を調べる．

謝辞

本報告書の作成にあたり，貴重なご指摘と熱心な御指導をいただいた山下信雄准教授に深く感謝の意を表します．また，日頃よりお世話になっている福嶋雅夫教授，林俊介助教，本研究を進める上で数多くの御助言をいただいた修士課程の黒川典俊さんをはじめとする福嶋研究室の皆様に厚く御礼申し上げます．

参考文献

[1] H. G. Beyer and D.B.Arnold, *Qualms regarding the optimality of cumulative path length control in CSA/CMA-evolution strategies*, Evolutionary Computation, **11** (2003), pp. 19–28.

[2] K. Deb, *An efficient constraint handling method for genetic algorithms*, Computer Methods in Applied Mechanics and Engineering, **186, issues 2–4** (2000), pp. 311–338.

[3] E. D. Dolan and J. J. More, *Benchmarking optimization software with performance profiles*, Mathematical programming, **91** (2002), pp. 201–213.

[4] M. L. F. Herrera and J. L. Verdegay, *Tackling real-coded genetic algorithms:oparators and tools for behavioural analysis*, Aritificial Intelligence Review, **12** (1998), pp. 265–319.

[5] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Mathematical programming, **91** (2002), pp. 239–269.

[6] ——, *On the global convergence a FILTER-SQP algorithm*, SIAM Journal on Optimization, **13** (2002), pp. 44–59.

[7] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[8] N. Hansen, *The CMA evolution strategy:a tutorial*, (2005), http://www.bionik.tuberlin.de/user/niko/cmatutorial.pdf.

[9] A. Hedar, *Studies on metaheuristics for completely derandomized self-adaptation in evolution strategies*, Evolutionary Computation, **9** (2001), pp. 159–195.

[10] A. R. Hedar, *Studies on metaheuristics for continuous global optimization problems*, Ph. D. thesis, Kyoto University, (2004).

[11] A. R. Hedar and M. Fukushima, *Derivative-free filter simulated annealing method for constrained continuous global optimization*, Journal of Global Optimization, **35** (2006), pp. 521–549.

[12] C. T. Kelley, *Detection and remediation of stagnation in the nelder-mead aogorithm using a sufficient decrease condition*, SIAM Journal on Optimization, **10** (1999), pp. 43–55.

[13] S. M. N. Hansen and P. Koumoutsakos, *Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation(cma-es)*, Evolutionary Computation, **11** (2003), pp. 1–18.

[14] D. O. N. I. M. Gould and P. L. Toint, *Cuter and sifdec: A constrained and unconstrained testing environment, revisited*, ACM Transactions on Mathematical Software, **20** (2003), pp. 373–394.

[15] 三宮 信雄, 玉置 久, 岩本 貴司，遺伝アルゴリズムと最適化, 朝倉書店, 1998.

付録 A　CMA 進化戦略

A.1　探索点の生成

CMA 進化戦略では，各反復 (世代という) において，新規の探索点を n 次元の多変量正規分布を用いて生成する．各世代で新規の探索点を λ 個 ($\lambda \geq 2$, 固定) 生成するとき，世代 $g+1$ における探索点 $x_k^{(g+1)} \in \mathbb{R}^n$, $k = 1, \ldots, \lambda$ は，世代 g における探索点の重みつき平均 $m^{(g)}$，重みつき共分散行列 $C^{(g)}$, およびパラメータ $\sigma^{(g)}$ を用いて,

$$x_k^{(g+1)} \sim \mathcal{N}\left(m^{(g)}, \left(\sigma^{(g)}\right)^2 C^{(g)}\right) \quad \text{for } k = 1, \ldots, \lambda \tag{1}$$

を満たすように生成される．ただし，$\mathcal{N}(m^{(g)}, (\sigma^{(g)})^2 C^{(g)})$ は平均 $m^{(g)}$, 分散 $(\sigma^{(g)})^2 C^{(g)}$ の n 次元多変量正規分布を表す.

各世代における探索点の重みつき平均 $m^{(g)}$，重みつき共分散行列 $C^{(g)}$，およびパラメータ $\sigma^{(g)}$ の更新方法を以下で述べる.

A.2　平均の更新

世代 g における λ 個の探索点 $x_1^{(g+1)}, \ldots, x_\lambda^{(g+1)}$ から, その点における目的関数 f の値の小さい順に μ 個 ($\mu \leq \lambda$，固定) の点を選び，それを $x_{1:\lambda}^{(g+1)}, \ldots, x_{\mu:\lambda}^{(g+1)}$ とする．($x_{i:\lambda}^{(g+1)}$ は探索点 $x_1^{(g+1)}, \ldots, x_\lambda^{(g+1)}$ のうち目的関数 f の値が i 番目に小さい点を表す．すなわち $f(x_{1:\lambda}^{(g+1)}) \leq f(x_{2:\lambda}^{(g+1)}) \leq \cdots \leq f(x_{\lambda:\lambda}^{(g+1)})$ を満たす．)

選択された μ 個の探索点 $x_{1:\lambda}^{(g+1)}, \ldots, x_{\mu:\lambda}^{(g+1)}$ を用いて，次の世代 $g+1$ における探索点の分布の平均 $m^{(g+1)}$ を定める．その際，選択された μ 個の探索点の単純な平均値から定めるのではなく，より目的関数 f の値の小さい点に重みをかけた重み付き平均を考える．μ 個の点の重み付き平均を次のように定める.

$$m^{(g+1)} = \sum_{i=1}^{\mu} \omega_i x_{i:\lambda}^{(g+1)} \tag{2}$$

ただし $\omega_i (i = 1 \ldots \mu)$ は結合のための正の重さ係数であり,

$$\sum_{i=1}^{\mu} \omega_i = 1, \qquad \omega_1 \geq \omega_2 \geq \cdots \geq \omega_\mu > 0 \tag{3}$$

を満たすようにとる．本報告書においては $\mu \approx \lambda/2$ とし

$$\mu = \left(\sum_{i=1}^{\mu} \omega_i^2\right)^{-1} \tag{4}$$

となるように重み係数 ω_i を定める.

A.3　共分散行列の更新

平均の更新と同様に目的関数 f の値の小さい μ 個の点 $x_{1:\lambda}^{(g+1)}, \ldots, x_{\mu:\lambda}^{(g+1)}$ を用いて共分散行列を更新する．共分散行列の更新式は以下のとおりである.

$$C^{(g+1)} = (1-c_{\mathrm{cov}})C^{(g)} + c_{\mathrm{cov}}\left(1-\frac{1}{\mu_{\mathrm{cov}}}\right)\sum_{i=1}^{\mu}\omega_i\left(\frac{x_{i:\lambda}^{(g+1)}-m^{(g)}}{\sigma^{(g)}}\right)\left(\frac{x_{i:\lambda}^{(g+1)}-m^{(g)}}{\sigma^{(g)}}\right)^T + \frac{c_{\mathrm{cov}}}{\mu_{\mathrm{cov}}}p_c^{(g+1)}{p_c^{(g+1)}}^T \tag{5}$$

第 1 項は過去の共分散行列の情報を使うための項であり．過去の情報と現代の情報の重みを $c_{\mathrm{cov}}(0 < c_{\mathrm{cov}} \leq 1)$ によって決定する．集合の大きさ λ を小さくすることで探索を早くすることができるが，そうすると良い共分散行列が得られない．過去の情報を用いることで集合の大きさ λ を小さくしても，良い共分散行列を推定することができる．

ここで第 2 項と第 3 項の重みを μ_{cov} によって決める．

第 2 項は現世代の選択された μ 個の点を用いた更新法である．この項によって得られる共分散行列は現世代において選択された点 $x_{i:\lambda}^{(g+1)}(i=1,\ldots,\mu)$ の分散ではなく，現世代の平均 $m^{(g)}$ から選択された点 $x_{i:\lambda}^{(g+1)}(i=1,\ldots,\mu)$ までのステップを $\sigma^{(g)}$ によって標準化した $(x_{i:\lambda}^{(g+1)}-m^{(g)})/\sigma^{(g)}$ の分散である．ここでも平均の更新と同様に単純な分散ではなく，より目的関数 f の値の小さい点に大きな重みをかける重みつき分散を考える．

第 3 項は平均 $m^{(g)}$ の推移の情報を用いた更新法であり，μ が小さい時に有効である．ここで用いられる $p_c^{(g)}$ は過去の平均 $m^{(g)}$ の推移の情報と現世代における平均の変化の情報の重みを表す $c_c \leq 1$ を用いて以下のように表されるものである．

$$p_c^{(g+1)} = (1-c_c)p_c^{(g)} + \alpha\frac{m^{(g+1)}-m^{(g)}}{\sigma^{(g)}} \tag{6}$$

α は p_c の正規化定数と呼ばる定数である．

ここで第 2 項は大きな母集団で重要であり，第 3 項は特に小さな母集団で重要である．

A.4 パラメータ σ の更新

共分散行列の相対的な大きさを表すパラメータ σ を更新する．このことによって収束の速さを高め，精度を向上させることができる．ここでも共分散行列の更新で用いたようには平均 $m^{(g)}$ の推移の情報を用いる．

平均の推移に正の相関関係にある，すなわち平均が同じような方向に動く傾向があるのであれば σ を大きくし，より広い範囲を探索できるようにする．逆に，平均の推移に負の相関関係にある，すなわち平均の動き方に互いに打ち消しあうような傾向があれば σ を小さくし，より狭い範囲を重点的に探索できるようにする．

以下のように p_σ を定義し，σ を更新する．

$$p_\sigma^{(g+1)} = (1-c_\sigma)p_\sigma^{(g)} + \beta\Big(C^{(g)}\Big)^{-\frac{1}{2}}\frac{m^{(g+1)}-m^{(g)}}{\sigma^{(g)}} \tag{7}$$

$$\sigma^{(g+1)} = \sigma^{(g)}\exp\left(c_\sigma\left(\frac{\|p_\sigma^{(g+1)}\|}{\sqrt{n}}-1\right)\right) \tag{8}$$

ただし

$c_\sigma < 1$ は前世代までの平均の推移と現世代での平均の推移の重みを表す定数で，β は p_c の正規化定数と呼ばれ，現世代でのステップが無相関であれば $\|p_\sigma^{(g+1)}\| = \sqrt{n}$ となるように選ばれる．

A.5 アルゴリズムの概要

CMA 進化戦略のアルゴリズムの流れをまとめると，次のようになる．

Step 0（パラメータの設定）

各世代で生成する探索点数 λ, 選択する点の数 μ, 重み $\omega_i (i = 1, \ldots, \mu)$, および c_σ,d_σ,c_c,μ_{COV},c_{COV} を設定する．

Step 1（初期化）

世代 $g = 0$ とする．初期共分散 $C^{(0)} = I$ とする．$\sigma^{(0)} = 0.5$ とする．初期平均 $m^{(0)} \in \mathbb{R}^n$ を問題に応じて決める．

Step 2（探索点の生成）

平均 $m^{(g)}$，共分散行列 $C^{(g)}$，$\sigma^{(g)}$ を用いて，式 (1) に従って探索点を作成する．

Step 3（探索点の更新）

Step 3-1: Step 2 で生成した探索点に順位をつける．

Step 3-2: 順位の高い方から μ 個の探索点を選択する．

Step 3-3: 式 (2) に従い，$m^{(g+1)}$ を求める．

Step 3-4: 式 (5) により，$C^{(g+1)}$ を更新する．

Step 3-5: 式 (8) により，$\sigma^{(g+1)}$ を更新する．

Step 4（停止条件のチェック）

停止条件を満たしていれば，最適値として $x_1^{(g+1)}$ を出力して終了する．そうでなければ，$g = g + 1$ として Step 2 に戻る