多目的遺伝的アルゴリズムによる空力最適設計

東北大学大学院工学研究科
航空宇宙工学専攻
大林　茂

1．緒　言

流体問題における最適化は、長い歴史を持ちまた広範な内容を含む課題である。人類の農耕文明の発生以来、水路の設計は常に重要な課題であったろうし、近年でもビルの空調から航空機の制御にいたるまで、流体問題に設計・制御・最適化が必要とされる話題には事欠かない。しかし、このような問題は、これまで主に流体の支配方程式を解くことなしに（しかもしばしば適切に）処理されてきた。たとえば航空機の最適制御理論は、流体の方程式と連立して解かれるのではなく、前もって与えられた空力係数から外力を評価する機体の運動方程式に基づいて構築されており、それでも航空機は(比較的)安全に飛んでいる。

一方、流体力学の理論的発展に伴い、超音速から亜音速へ衝撃波を伴わずに減速する遷音速翼型の設計や吸い込み・吹き出しによる境界層制御など、高度の流体力学の知識を駆使した設計・制御・最適化も見られるようになった。このような研究は、最適化の理論を用いることなく流体の物理的性質を調べることで進められてきた。たとえば遷音速翼型で、超音速から亜音速への減速時に衝撃波を回避できれば造波抵抗はほぼ零となり、そのような翼型は衝撃波を伴う通常の翼型に比べはるかに抵抗が小さい。しかもそのような翼型の存在する条件が特殊なため、一つでも見つけられればそれが最適解であると見なされてきた。

このような「流体なしの最適制御」・「最適化なしの流体設計」に対して、流体の支配方程式に基づく最適化問題の定式化を考えることができる。それが役に立つかは別にして、このような問題の定式化がこれまでされてこなかったのは、流体の方程式を解くのが大変だったためである。現在では、計算流体力学（CFD）によって流体の支配方程式が数値的に解けるようになってきたので、流体の支配方程式に基づく最適化が注目されている。またこのような機運を受けて、CFD と最適化法を組み合わせた研究が最近学会などでも発表されるようになってきた。もっとも、現在でもまだリアルタイムでの非定常流体力の計算は大変なので、流体問題の最適化ではオフラインで行える形状最適化にもっぱら関心が集まっている。

一口に最適化といっても、流体問題の最適化は非常に複雑な問題である。揚力や抵抗といった流体力学的な性能は、そもそも非線形な関数であり、しかも形状変化に対して非常に敏感である。流体問題の最適化では、そのような関数を目的関数として最適化すること自体が大変なうえに、流体の支配方程式自体を満たすことが最初の制約条件となるからである。

目的関数の複雑さは、局所的な最適解の多さにも現れる。たとえていうならば、本州に下関から上陸して一つ一つ山登りをして、最も高い山である富士山を見つけられるかという問題である。ここで最適化の戦略は大きく二つの場合に分かれる。

富士山を見つけるのは大変だけどとりあえず身近に高い山があれば十分な場合と、あくまで富士山を見つけたい場合である。前者の場合、最適化の初期値をこれまでの経験から十分に最適解に近くとれるならば、勾配法などを用いて効率的に最適解を見つける工夫をすることができる。一方後者の場合は、確実に大域的な最適解を見つける方法はないので、確率的な方法や進化的な方法に頼ることになる。本稿では、大域的な最適解を求める立場から進化的な計算法に注目して、流体問題の最適化の特徴やひとつの最適化問題からより多くの情報を引き出す方法について考察してみよう。

２．空力問題における最適化法の比較

制約条件を満たすものの中で、目的関数を最大あるいは最小にするような設計変数の値を決定する、という形に定式化される問題を最適化問題という。ここで、設計変数とは設計を決定する主要素である未知のパラメータのことであり、また、設計上の安定性・有用性・機能性を保持するために、設計変数やその他の状態変数に課せられる条件を制約（拘束）条件という。普通、制約条件を満足する設計は無数に存在するが、その中から何らかの基準を設けて最終的に 1 つの設計に決定する必要があり、その基準を見積もるための関数を導入しなければならない。これを目的(評価)関数という。一般に制約条件は等式および不等式で表され、設計変数は多変数の場合がほとんどなので設計変数ベクトルとして扱われる。

最適化問題の定式化の形としては

$$\text{目的関数}\ :\ F(\mathbf{X}) \rightarrow \text{最大（or 最小）} \tag{1}$$

$$\text{制約条件}\ :\ \begin{array}{ll} g_j(\mathbf{X}) \le 0 & (j = 1,2,\cdots,k) \\ h_j(\mathbf{X}) = 0 & (j = k+1,k+2,\cdots,l) \end{array} \tag{2}$$

$$\text{設計変数ベクトル：}\ \mathbf{X} = (x_1, x_2, \cdots, x_n) \tag{3}$$

となる。

この最適化問題をいかに解くかを考えよう。現在では様々な最適化手法が考案されているが、いったいどのような方法があり、どのような特徴があるのかを知りたい。そこで、本節では代表的な最適化法として、勾配法（The gradient-based method, GM）[1]、焼きなまし法（Simulated Annealing, SA）[2]、そして遺伝的アルゴリズム（Genetic Algorithm, GA）[3]の３つを取り上げて概説し、実際の最適化問題を解くことによって３つの手法を比較する。

典型的な空力問題として、翼型の形状最適化を考えよう。一般に形状最適化問題において最適化すべき形状をどのように定義するか、つまり設計変数をどのように設定するかは非常に重要なポイントとなる。翼型の表現方法には、大きくわけて

(1) 点列による直接定義
(2) 基底関数の組み合わせ（多項式やスプライン等）
(3) 基底翼型の線形結合[4]
(4) 基本翼型と摂動関数の組み合わせ[5]

の 4 種類がある。特に、翼型の空力性能は微小な形状変化に対して敏感であり、一

般的な高精度の形状定義を行うためには多くの設計変数を必要とする。また、設計変数の個数は最適解を探索する設計空間の次数を決める。従って、設計変数が多すぎると設計空間の次数も高くなるので、最適解の探索が非常に困難になる。このため、設計変数はできるだけ少ないほうが望ましい。そこで、(1)や(2)のような一般的な形状定義を行う代わりに、(3)や(4)のように既存の設計例のいくつかをもとにして、それらの線形結合や摂動で最適化すべき翼型形状を表す近似的最適化手法がよく用いられる。これらの手法では設計変数を極端に減らせる一方、設計変数の定義域を負から正まで幅広くとれば、かなり自由度のある翼型形状が表現可能である。本章では、Vanderplaats により提案された(3)の方法を採用する。また、ここで取り上げた最適化問題における目的関数がどのように分布しているかを示すことにより、３つの最適化手法の適用性を理解する。

2.1 勾配法（The gradient-based method, GM）

最適化を考える上で最も代表的な方法が勾配法である。目的関数の最大化を考えるとき、この方法は山登り法（Hill-climbing strategy）ともいうべき単純な考え方に基づいている。すなわち、ある地点から山の頂上にたどり着くためには、その場での勾配の一番きつい方向に登っていけばよい[1]。一般的には次式に従い、繰り返し計算により最適解を得る。

$$\mathbf{X}^{q+1} = \mathbf{X}^{q} + \alpha \cdot \mathbf{S}^{q} \tag{4}$$

ここで、**X**は設計変数ベクトル、**S**は探索方向ベクトル、αはステップ幅、qは試行回数を表している。勾配法の一般的なフローチャートを図 1 に示す。この図から実際の最適化計算では３つの要素から成り立っていることが分かる。

1. 探索方向**S**を決める。
2. ステップ幅αを決めるために一次元探索を実行する。
3. 収束判定を行う。

探索方向 S の決定

普通、探索方向**S**には目的関数$F(\mathbf{X})$の勾配$\nabla F(\mathbf{X})$が用いられる（最大化問題では$\mathbf{S} = \nabla F(\mathbf{X})$、最小化問題では$\mathbf{S} = -\nabla F(\mathbf{X})$）。ところが、探索方向に制約条件という壁がある場合にはこれ以上進むことができなくなってしまうので、探索方向を変えなければならない。また、上手く方向を変えないと最適解から離れてしまう恐れがある。そこで、このときの探索方向を決める方法として実行可能方向法（Feasible Directions Method）[1]を採用する。例えば、最適化問題が以下のように定式化された場合を考えてみよう。

$$\text{目的関数：} F(\mathbf{X}) \rightarrow \text{最大化} \tag{5}$$

$$\text{制約条件：} g_1(\mathbf{X}) \le 0 \tag{6}$$

$$\text{設計変数ベクトル：} \mathbf{X} = (x_1, x_2) \tag{7}$$

ここで、制約条件$g_1(\mathbf{X})$の境界上に現在の設計点$\mathbf{X}^1$があるとする。このときの探索方向**S**を決定するには、

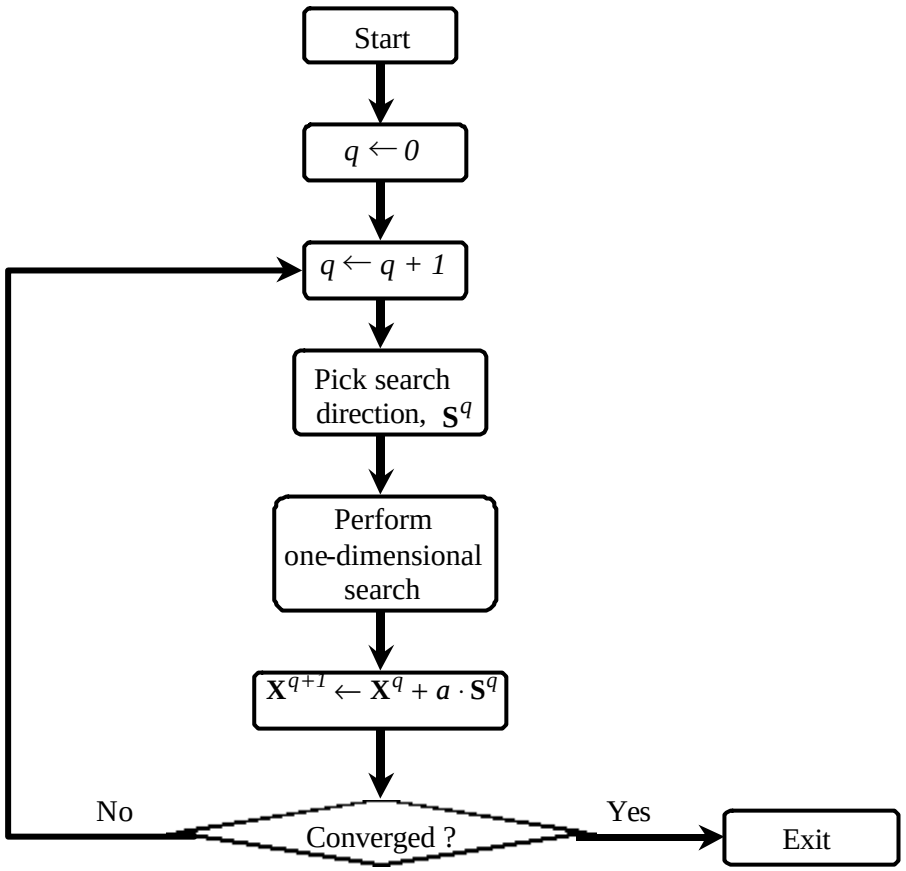

図 1　一般的な勾配法

1. **設計点 $\mathbf{X}^1$ での勾配（$\nabla F(\mathbf{X}^1)$, $\nabla g_1(\mathbf{X}^1)$）を求める。**
2. **次の 2 つの不等式を満たすような探索方向 S を求める。**

$$\nabla F(\mathbf{X}^1)\cdot\mathbf{S} \geq 0 \tag{8}$$

$$\nabla g_1(\mathbf{X}^1)\cdot\mathbf{S} \leq 0 \tag{9}$$

の操作を行えばよい。ここで、(8)式を満たす領域を *Usable sector*、探索方向を *Usable direction*、式(9)を満たす領域を *Feasible sector*、探索方向を *Feasible direction* という。この関係を図 2 に示す。このとき、式(9)を満たす S のなかで、式(8)を最大にするような S を選べば、目的関数 $F(\mathbf{X})$ を最も増加させることができる。この S を *Usable-feasible direction* という。

ステップ幅 α を決めるための一次元探索

探索方向が決定したら、今度は次の設計点までどの程度移動すべきか、そのステップ幅 α を決定しなければばならない。式(4)において、現在の設計点 $\mathbf{X}^q$ と探索方向 S は既知であるので、α が分かれば次の設計点 $\mathbf{X}^{q+1}$ が決まる。つまり、スカラー量である α のみを決定する一次元探索問題となる。

そこで、例として定式化された上の最適化問題は最大化を考えているので、次のような一変数最大化問題を解けばよい。

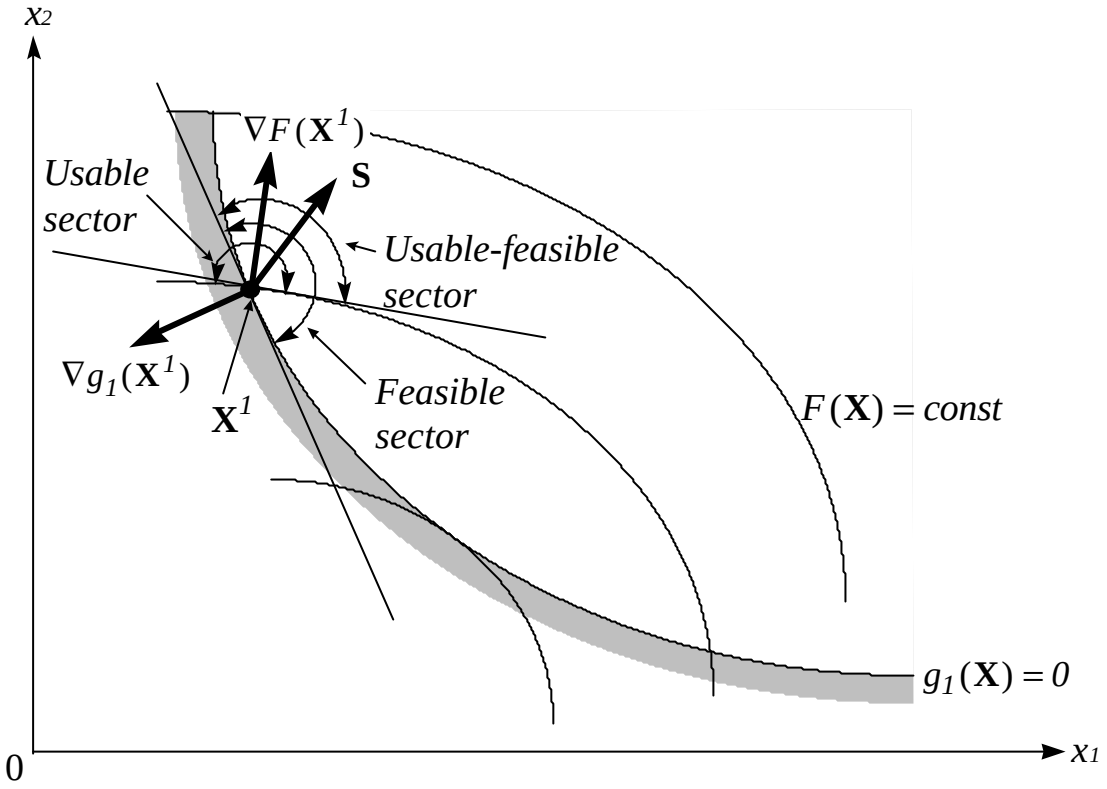

図 2　*Usable-feasible* **探索方向 S**

$$\text{目的関数：}\ F(\mathbf{X}^q + \alpha \cdot \mathbf{S}^q) = F(\alpha) \rightarrow \text{最大化} \tag{10}$$

$$\text{制約条件：}\ g_1(\mathbf{X}^q + \alpha \cdot \mathbf{S}^q) = g_1(\alpha) \le 0 \tag{11}$$

ここでは、この問題を解くために黄金分割法[1]**を適用した。ある区間内に二つの試行点を設け、この２点の関数値の比較により最適解の存在範囲を限定する。このような操作を新たに定まる区間に対して繰り返し適用していけば、最適解の存在範囲をいくらでも小さくすることができる。二つの試行点を設けたことで、必ずどちらか１点が区間内に入るので、その点をそのまま次の反復でも試行点として利用でき、各反復で一つの施行点に対する関数値を新たに計算するだけでよい。このような考え方を用いた代表的な方法が黄金分割法である。そのアルゴリズムを図** 3 **に示す。ここで、** x_u, x_ℓ, x_1, x_2 **はそれぞれ、区間の上限、下限、試行点** $(x_1 < x_2)$ **を表す。**

収束判定

全般的な最適化過程の収束を判定する基準として３つ考えられる。

(1)**最大繰り返し数**
(2)**目的関数の絶対的または相対的な変化**
(3)Kuhn-Tucker **条件**

(1)**は、最大繰り返し数に達すると強制的にアルゴリズムを終了させるもの。**(2)**では、次のように** ε **を正の小さい数としたときに式**(12)**あるいは式**(13)**を満たした場合に収束したものと見なしている。**

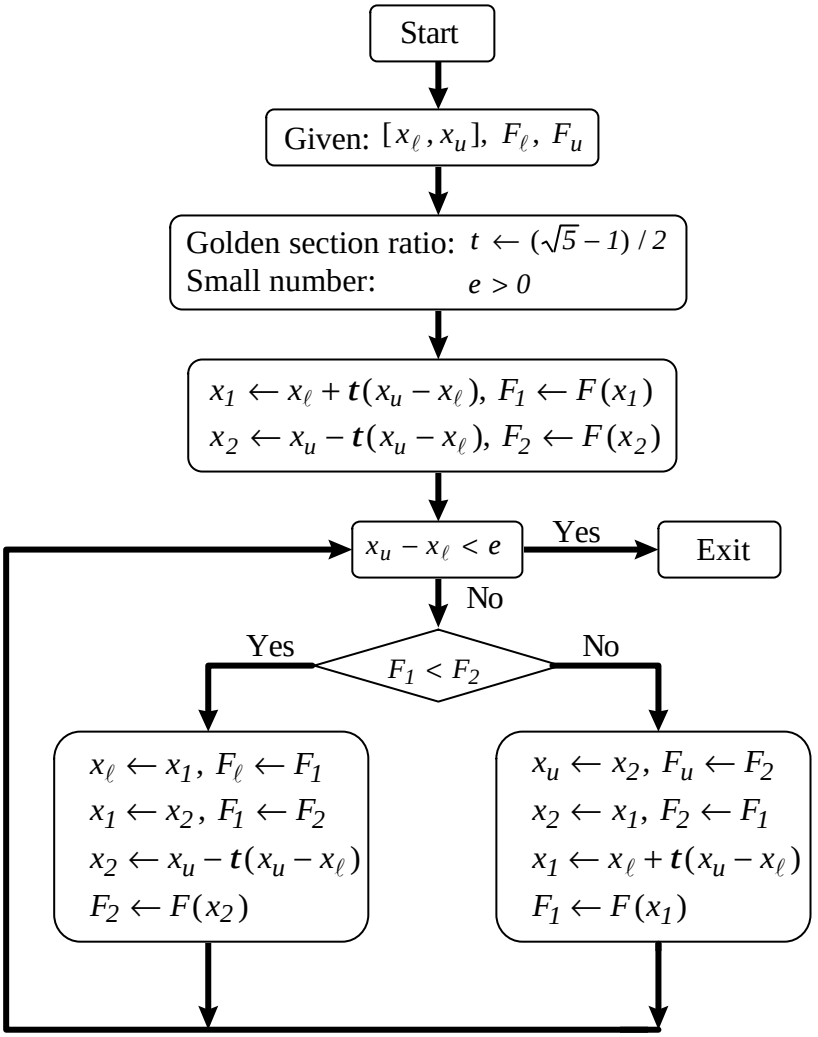

図 3　黄金分割法のアルゴリズム

$$\left|F(\mathbf{X}^q) - F(\mathbf{X}^{q-1})\right| \le e \tag{12}$$

$$\frac{\left|F(\mathbf{X}^q) - F(\mathbf{X}^{q-1})\right|}{\max\left[\left|F(\mathbf{X}^q)\right|, 10^{-10}\right]} \le e \tag{13}$$

(13)は、簡潔にいうと $F(\mathbf{X}^q)$ の勾配が $\nabla F(\mathbf{X}^q) \le e$ となることを意味している。この基準は最適性に対する必要条件であり、勾配法にはよく用いられる。

以上、勾配法の計算を構成する３つの要素について説明した。もちろん、どの要素も単峰性、つまり山の頂上が１つしかないということを仮定している。もし山が１つしかないことが保証されれば、この方法で必ず頂上にたどり着けるはずである。すなわち、目的関数が微分可能かつ凸関数であれば、勾配法で求められた解は大域的最適解であることが分かっている。しかし現実的には単峰性の保証はなく、勾配法は初期設計のごく近傍で局所的な最適解を求めているにすぎない。このため、こ

の方法を適用するときは初期値を変えて、いくつもの局所的最適解を求め、その中から一番良いものを選ぶという(あまり理論的でない)操作が必要になる。

2.2 焼きなまし法（Simulated Annealing, SA）

SA は、物理現象をアナロジーした代表的かつ有用的な最適化手法であり、統計物理学で用いられている焼きなましとの類比から考案された。つまり、結晶性の物質（例えば金属）を十分加熱して液体状態にした後、ゆっくりと十分時間をかけて冷やすと、エネルギが最小状態になる分子配列となり純結晶状態に落ち着く。これを統計力学的に考えると、現在の温度 T の状態 i でのエネルギ E_i とし、分子運動などによって起こりうる次の状態 j でのエネルギ E_j とするとき、エネルギ差 $\Delta E = E_j - E_i$ が $\Delta E < 0$ ならばエネルギ的に安定な状態 j に推移するとみられる。一方、$\Delta E > 0$ であっても、次のようなボルツマンの確率分布

$$\mathrm{Prob}(\Delta E) = \exp\left(\frac{-\Delta E}{kT}\right) \tag{14}$$

に従って状態 j に推移できるとする[2]。ここで、ボルツマン定数 k は自然界で温度とエネルギを関連づける定数である。

このような考え方をもとにすると、SA はエネルギを最小化する目的関数に対応させたアルゴリズムであるといえる。最適解を探索する繰り返し式(4)について考えてみると、SA では $\mathbf{X}^q$ に適当な擾乱 $\Delta\mathbf{X}^q$ を加え、目的関数が減る場合には無条件に、増える場合には式(14)で与えられる確率で、$\mathbf{X}^{q+1}$ として設計を更新する。このとき、単に目的関数の値が減る場合だけを受け入れるなら、山登り法の逆をしているだけで局所解しか探索できないことになる。ところが、式(14)のボルツマン分布に従う確率で悪い解（目的関数が増える解）も受け入れることで大域的な最適解の探索も期待できる。図 4 に SA のアルゴリズムを示す。

ここで、SA の収束判定について考える。どんな最適化アルゴリズムも収束判定は微妙である。第一に、目的関数の最大評価回数を設定することが挙げられる。これは、収束判定というよりもむしろ計算を終了させる限界値であり、計算が無限に繰り返されることを防いでいることになる。第二に、目的関数値の減少の相対値が、ある小さい相対許容値（普通、機械精度の程度）より小さくなったときに終了させる場合もある。これが実質的な収束判定の基準といえる。本研究では両者に対して収束判定を行う。つまり、後者の収束判定を優先的に考え、この条件を満たす場合は、目的関数の評価回数が最大評価回数に達していなくともアルゴリズムを終了させ、最大評価回数に達したあとでも依然として後者の条件を満たさない場合は強制的に終了させることにする。

SA は統計力学との類比から、ある仮定のもとで最適解に漸近的に収束することが証明されている。しかし、有限の試行回数でより早く近似的な最適解を得るには、設計変数に対する擾乱の入れ方や、温度を徐々に下げる冷却過程（クーリングスケジュール）の制御が重要であり、これらは非常に難しい。というのは、温度を冷やすと式(14) によって与えられる確率、すなわち局所解から脱出できる確率が低くなるからである。従って、勾配法と同様にいくつかの初期値から最適解を求め、比較し、その中から最も良いものを選ぶことになる。あるいは、計算の終了した点を再び初

期値に設定してリスタートさせ、これを何回か繰り返し、最終的に得られた解を最適解とする場合も考えられる。

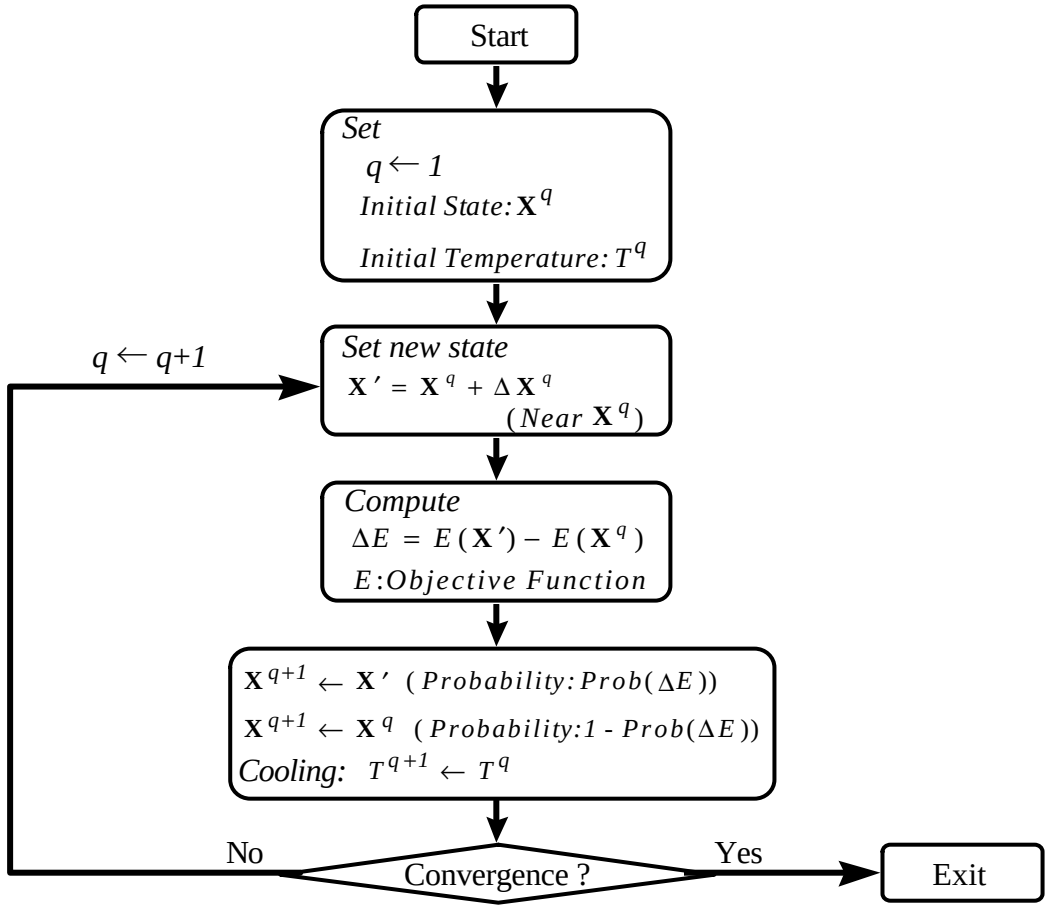

図 4　焼きなまし法（SA）のアルゴリズム

Downhill Simplex ***法***

また SA **では、擾乱** $\Delta\mathbf{X}^q$ **をどのように与えるのかという問題も重要である。この問題については、**Downhill Simplex **法と組み合わせる手法が提案されている**[2]**。ここで、**Downhill Simplex **法の考え方を合わせて紹介しておこう。***N* **次元空間における** *N* +1 **角形を** simplex**（単体）と呼ぶ。設計変数の次元を** *N* **とするとき、** *N* +1 **回の試行（目的関数評価）を行う、すなわちこの単体を計算することで、目的関数の勾配情報を得ることができる。逆に勾配情報を計算するには、最低限** *N* +1 **回の試行を行う必要がある。図** 5 **に** 2 **次元の設計空間における単体すなわち三角形を図示する。**

Downhill Simplex **法では、**simplex **の定義する勾配方向に次々と** simplex **を折り返して坂を下り、目的関数の最小値を探索する。この手続きを** 2 **次元で図示すると、図** 6 **のようになる。図中の点線は目的関数の等高線で、右上に行くほど関数値が低くなっている様子を示している。最適化のプロセスは左下の三角形から始まり、これを坂を下る方向に折り返すことで右上の最適解の方へ近づいていく。いわゆる勾配法のように微分を必要とせず、単純に谷（あるいは山）を探していく方法であることが分かる。**

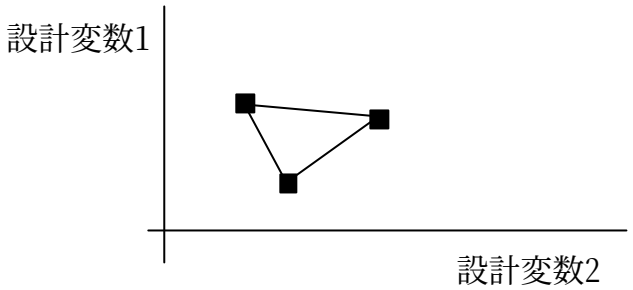

図 5　2 設計変数に対する 3 試行で作られる Simplex（単体）

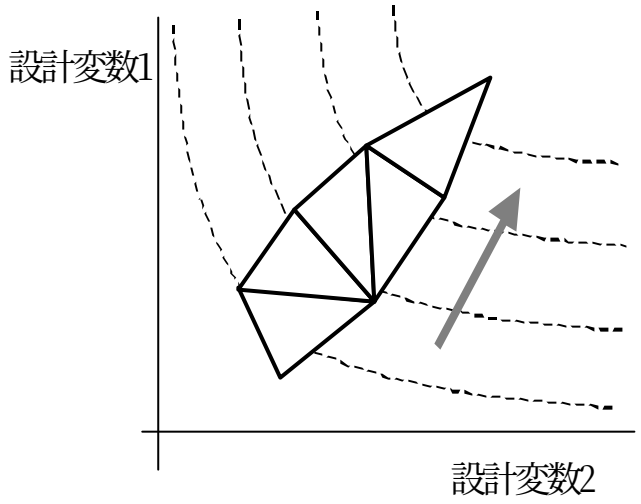

図 6　目的関数の勾配

この方法を実際に用いる場合は、同じところをぐるぐる回ったりしないように、さらにいくつかの単純なルールを付け加える必要がある。特に、探索を効率的にするためには、simplex の拡大・縮小を取り入れる必要がある。これらについては文献[2]を参考にしていただきたい。

また、最大値を求めたいなら当然 uphill を繰り返す。この方法は線形計画法でいうところのシンプレクス法ではない。誤解を避けるために面倒でも downhill なり uphill なりをつけて、たとえば Downhill Simplex 法のように呼んでいる。

この方法では、各ステップで最も悪い点を捨てて新しい点を付け加えることで、新たな simplex を作成している。この新しい点を $\mathbf{X}^q + \Delta\mathbf{X}^q$ として SA に用いるハイブリッド法が、文献[2]で提案されている。局所的な最適値が多数ある場合には simplex を折り返しても関数値が必ず改善されるとはいえず、改善されなければ通常の Downhill Simplex 法はそこで停止してしまう。このとき、SA の定める確率で改悪された simplex も受け入れることでより大域的な最適解の探索が可能になる。この手法はクーリングスケジュールの制御にまだ問題を残しているが、とりあえず試してみるには十分な頑丈な方法である。

2.3　遺伝的アルゴリズム（Genetic Algorithm, GA）

メンデルの法則とダーウィンの理論で知られているように、すべての生物は生殖と淘汰および突然変異によって環境に適応しながら進化する。その生物進化の原理は、生物を構成している細胞中の核が染色体という形でそれら生物固有の遺伝プログラム（遺伝子）を持っており、これが生殖、淘汰、突然変異によって様々に変更され、次世代に引き継がれていくと言われている。GA は、このような生物進化の過程そのものをモデル化し、最適化問題の解法に応用したアルゴリズムである[3]。具体的には、“ 人口（Population）” と呼ばれる解の集団を作り、これを構成する “ 個体（Individual）” と呼ばれる解候補群が “ 選択（Selection）”、“ 交叉（Crossover）” そして “ 突然変異（Mutation）” という過程を繰り返しながら、最適解へと収束してゆくものである。図 7 に GA の基本的な動作の流れを示す。

GA は従来の最適化手法と、以下の点で大きく異なる特徴を持っている。

① 解の探索には、設計変数を２進数などにコード化した“ 染色体（Chromosome）” と呼ばれる遺伝子を用いる。
② 勾配法や SA のような１点からの探索（図 8）ではなくて、多数の点からの同時探索（図 9）を行う。
③ 解の評価には目的関数のみを用い、その勾配（微分値）は用いない。
④ 決定論的ではなく、確率的な方法である。

以上のような特徴から明らかなように、GA は与えられた問題の世界の知識がなくても、評価値のみが分かれば問題を解くことができ、また多点同時探索法であることから、多峰性の強い問題であってもその大域的最適解か、あるいはそれに近い解候補をいくつか探索することが可能である。ここで図 8 において、SA の場合は基本的に最小化問題を扱うが、ここでは山登りという観点から、勾配法、GA と同様に最大化問題に置き換え、その作業状況を図示した。図 8 中の *Transition* は、ボルツマンの確率分布式(14)に従った探索点の推移を表す。これにより、局所的な解からの脱出が可能となる。また、SA や GA では大域的な解に近づこうとするため探索の対象となる山が変わっていくが、勾配法では１つの山のみを探索しているため局所解に陥っている様子がわかる。

このような特徴から GA は様々な問題に対して適用することができるため、ロバスト性（多様な問題に対応でき、良い性能を示すことが期待できること）に優れたアルゴリズムであるといえよう。しかし、どのような条件で最適解に収束するのかは理論的に明らかではない。実際、交叉や突然変異を行ったあとで必ずしも良い方向に進むとは限らない（図 9）。一方、勾配法でいくつもの初期値から得られた局所的最適解から最も良いものを選ぶことを考えると、結局両者とも理論的とはいえない。むしろ、勾配法で初期値を変えるという任意性のある操作を、GA では確率論的探索という形でアルゴリズムに取り入れているといえる。また、GA は他の最適化手法に比べてかなり多くの計算時間を要する。これは、GA の特徴②より多くの探索点を用いているため生じてしまう欠点である。空力最適化では計算時間のほとんどが評価の部分に費やされているので、いかにして効率よく、速く評価するか、あるいは、GA の各オペレータを上手く工夫して最適解をいかに早く探索するが重要課題である。

次に、GA を用いる際の基本的な動作について説明する。

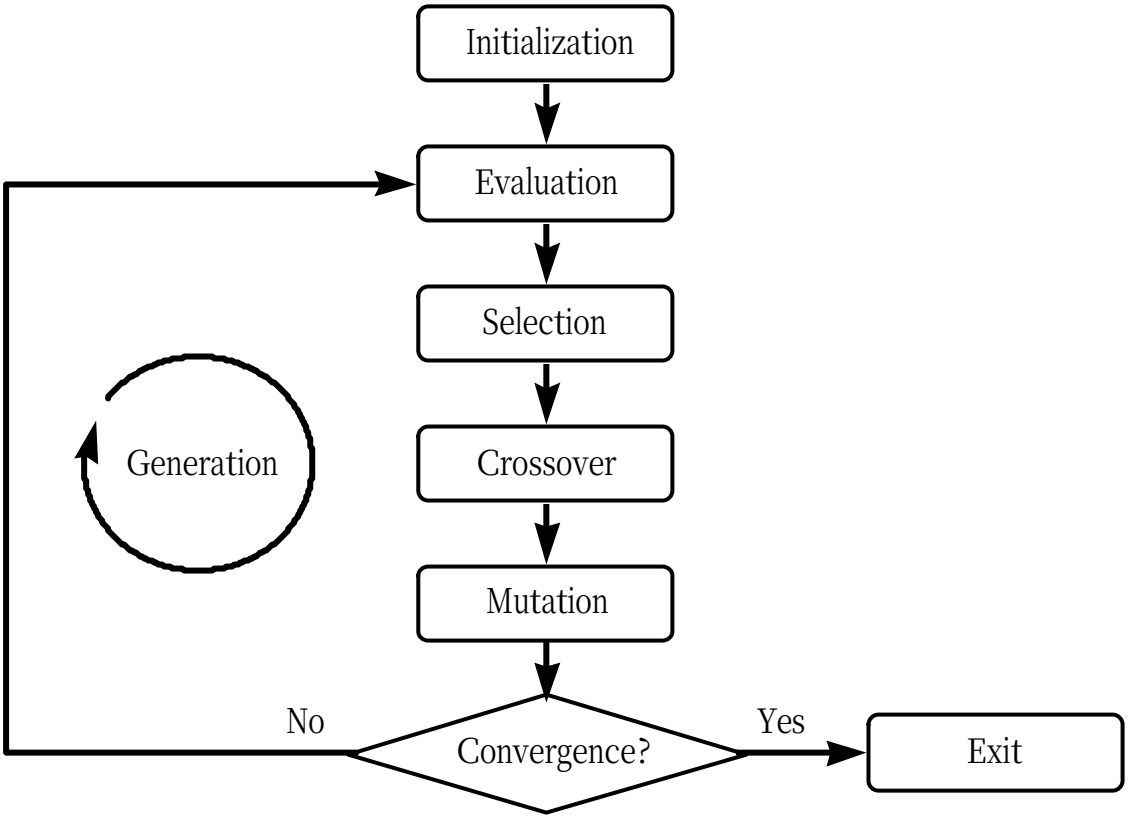

図 7　遺伝的アルゴリズム（GA）の基本的な動作

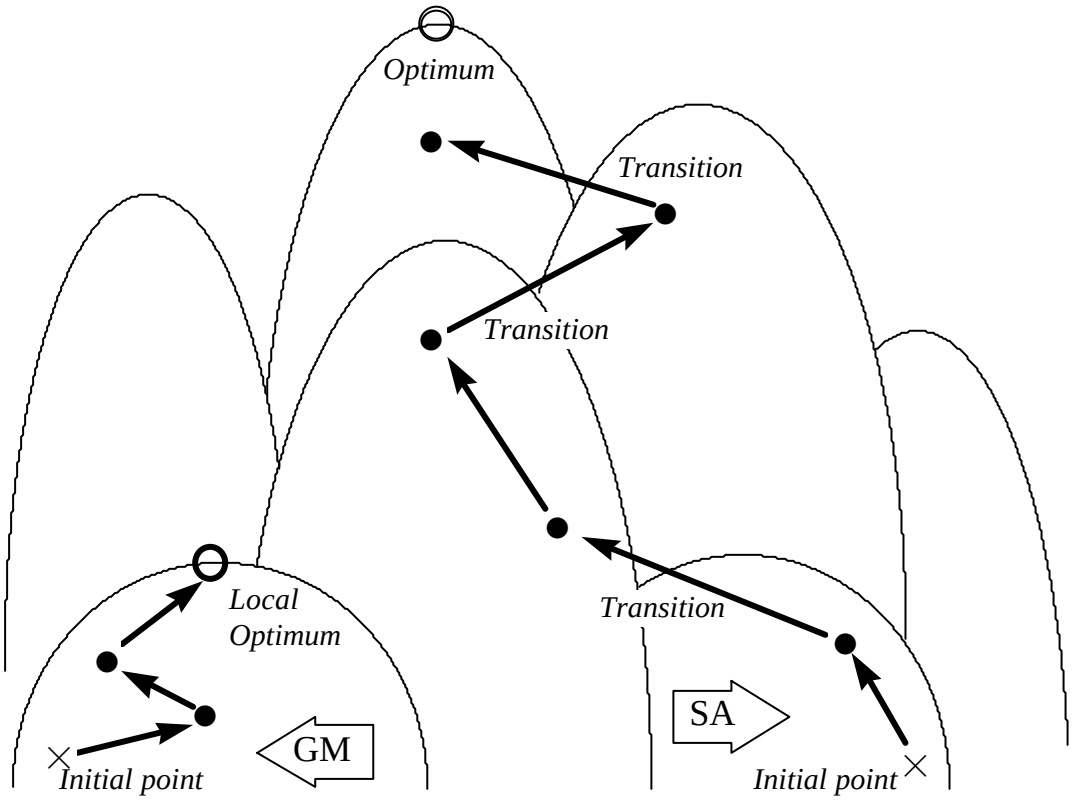

図 8　勾配法（GM）と SA の作業状況

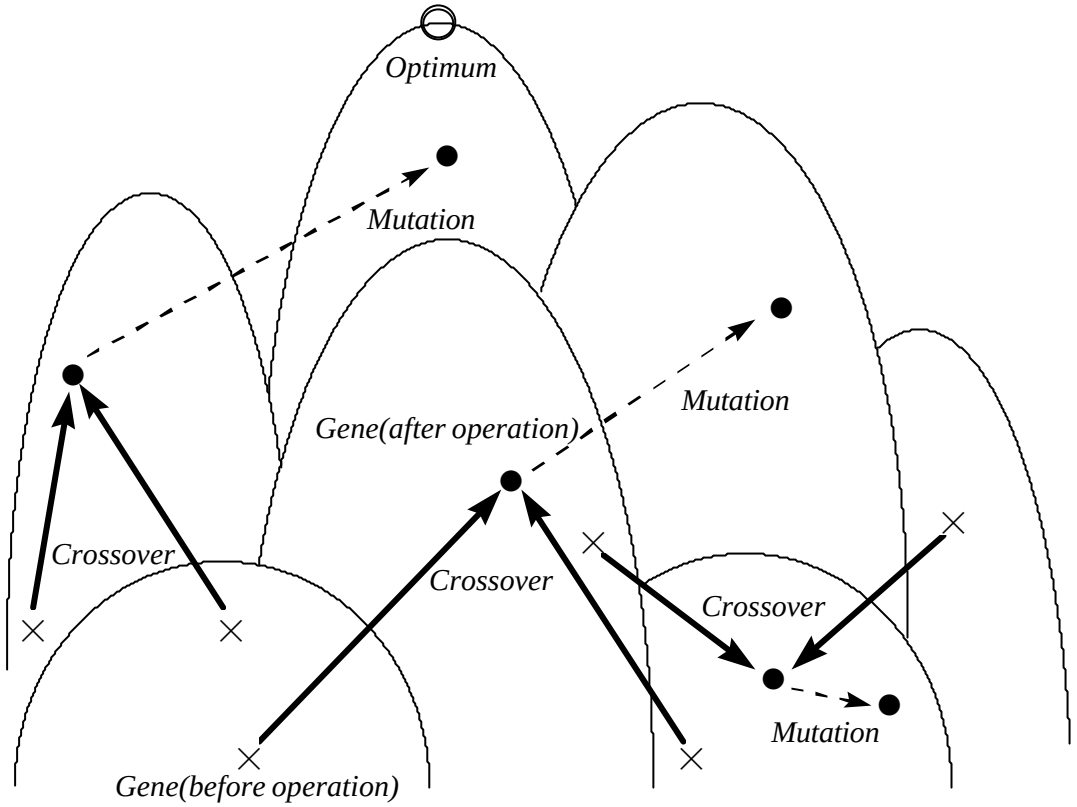

図 9　GA の作業状況

コード化とデコード化（Encoding・Decoding）

GA を用いて最適化問題を解くとき、初めに設計変数を遺伝子の形で表現しなければならない。この実際の設計変数を遺伝子型に変換することを“コード化”という。その逆の変換を“デコード化”という。どのようにコード化するかは問題によって異なる。というのは、交叉の部分では実際の個体ではなく、コード化された遺伝子を入れ換えることになるので、コード化が適切でないと解が収束しない、あるいは収束するまでに莫大な時間がかかるという危険性が生じるからである。従って、設計が成功するかどうかは設計者がどのようなコード化手法を選ぶかという設計者の能力に依存すると考えられる。そのため、コード化を行うときは適切に遺伝子を表現する必要がある。通常、遺伝子を表現するときは 2 値 {0, 1} の並びである文字列、つまりビット列（bit string）によるコード化が用いられる。ところが、実数を 2 進数に変換すると、精度が粗い、変化する量のスケールを GA オペレータで制御するのが困難である、などの問題が起こりやすい。そこで、物理的な意味での明確で効果的な遺伝的操作を行うために、実数型データをそのまま用いたり、区間[0,1]に正規化して用いたりして、遺伝子表現とする。

評価（Evaluation）

与えられた制約条件のもとで、集団中の各個体の優劣を決める。この評価の高い個体ほど、優れた個体として自分の遺伝子を次の世代に残すことのできる可能性が高いことになる。

空力最適化では流れ場を評価する必要があるので、CFD を用いることになる。進化的計算が終了するまでの全計算時間のうち、CFD の計算部分がその大半を占めるこ

とは、GA のプログラミングが比較的簡単であることからも理解できるであろう。従って、計算時間の短縮を考慮したり、一つの計算からより多くの情報を引き出す工夫をする必要がある。

選択（Selection）

集団中から実際に遺伝子の入れ換えを行う 2 つの個体（親）を選び出す。評価の良い個体間にできる子孫のほうが、評価の悪い個体間にできる子孫よりも優れている可能性が高いと考えられる。このため、選択を行うときは確率的に評価の高い個体が選ばれるようにする。選択手法として、ルーレット選択、ランキング選択、トーナメント選択などが提案されている[6]。

交叉（Crossover）

選択された個体間（親）で遺伝子の入れ換えを行い、新たな個体を形成し、この個体を子孫として次の世代に残す。ここで、入れ換える遺伝子の数が多すぎたり少なすぎたりすると、優れた子孫を残すことができなくなり、最適値の探索能力が衰えるので、どの程度遺伝子を入れ換えたらよいかという問題を慎重に考える必要がある。従って、交叉は GA で最も重要な役割を担っているといえる。

突然変異（Mutation）

交叉とは異なり、個体中の遺伝子を強制的に操作することによって、交叉だけでは得られないような解を探索するために用いられる。突然変異が起こる確率を突然変異率という。突然変異率が高すぎると、ランダムサーチのように解空間の中をランダムに探索するアルゴリズムとなってしまうので、解の収束性が落ちてしまう。このため、突然変異率は一般的に小さく設定されている。従って、GA では突然変異はあまり重要視されていないが、図 9 でも分かる通り、より大域的な解を探索するためにはある程度は必要である。

エリート戦略（Elite strategy）

GA では交叉、突然変異等のオペレータにより次の世代に残すべき子孫を形成するわけだが、それが親よりも優れた評価を持っているとは必ずしもいえない。反対に、形成された子孫の評価が悪くなり、一度発見された優秀な個体が生き残れずに悪い解に収束する可能性もありうる。こうした場合を避けるため、各世代の集団中で最も優れた個体は交叉、突然変異をすることなく優先的に次の世代に残すという手段をとる。これがエリート戦略である。

2.4　高揚力翼型の最適化

本節では、３つの最適化手法（勾配法・SA・GA）の空力最適化問題への適応性を比較するために、低速の高揚力翼型の形状最適化を行ってみよう。

定式化

翼型の空力特性は、その形状変化に対して非常に敏感に変わってしまう。このため、最適化を行うときに厳密に形状を定義しようとすると、パラメータとしての設計変数が極端に多くなり、解空間も大きくなるので最適解への収束が遅くなり、計算時間も増えてしまう。そこで、Vanderpplats は既知の翼型を基底翼型として、その線形結合により求める翼型を定義する方法を提案した[4]。この方法を用いると、形

状を直接定義する場合に比べてはるかに設計変数を減らすことができるため、最適化問題に取り入れることが可能になる。

本研究では、4 つの基底翼型（NACA2412, NACA64$_1$-412, NACA65$_2$-415, NACA64$_2$A215）（図10）を用いて、新しい翼型を

$$\mathbf{Y} = a_1\mathbf{Y}^1 + a_2\mathbf{Y}^2 + a_3\mathbf{Y}^3 + a_4\mathbf{Y}^4 \qquad (15)$$

と定義する。ここで、係数 $a_1 \sim a_4$ は設計変数、また $\mathbf{Y}^1 \sim \mathbf{Y}^4$ はそれぞれ NACA2412, NACA641-412, NACA652-415, NACA642A215 の与えられた x 座標での上下面の y 座標のベクトルである。つまり、既存の翼型の y 座標を基底関数として固定し、それぞれの翼型にかかる係数を変化させることにより最適翼型を得る。

最適化問題として、

C_ℓ（揚力係数）→最大化 (16)

制約条件：$t/c = 0.15$（最大翼厚比） (17)

定義域：$-5.0 < a_1 \sim a_4 < 5.0$（係数） (18)

を考える。式(18)では係数に負の領域も含めることによって、設計空間を単なる内挿でなくより広い空間にしている。また、流れ場は二次元非粘性・非圧縮性のポテンシャル流れを仮定し、数値計算法としてパネル法[7]を用いて解析する。パネル法は境界要素法の一種である。非圧縮流体の速度ポテンシャルφはラプラス方程式に従い、これを境界条件のもとで解くとφの一般解は積分表現となる。これは最終的に物体表面における吹き出しと二重吹き出しの２つの未知量を含む積分方程式になるが、一般にこの積分式を解析的に解くことは難しい。従って、近似的または数値的に解くことが行われる。このとき、物体表面を微小パネルに分割して離散化し、パネルごとに 2 つの未知量（吹き出しと二重吹き出し）を与えることにより連立一次方程式に帰着させることができる。これがパネル法である。この連立方程式を解くことによって各パネルの吹き出しと二重吹き出しが求まり、速度ポテンシャルから速度が分かる。従って、物体表面にかかる力も計算できる。今回は迎角は 6 度に設定し、パネル法を用いる時のパネル数は翼型の上・下面それぞれ 50 個、合計 100 個のパネルを使用している。

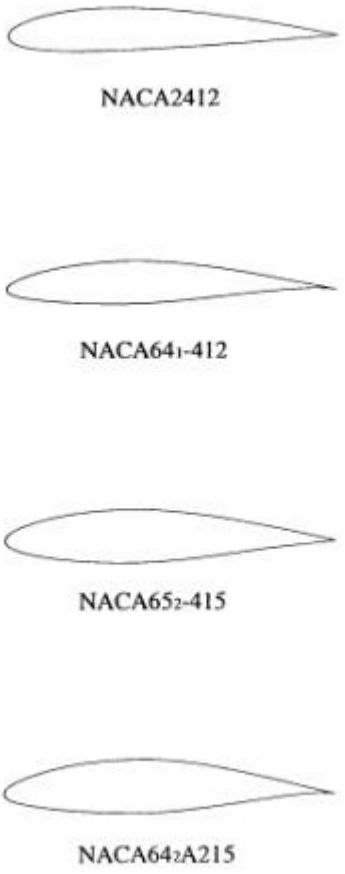

図 10　基底翼型

設計変数による評価関数の分布

ここでは、容易に評価関数の分布が表現できるように、2 設計変数 (a_1, a_2) のみを用いて（基底翼型は NACA2412 と NACA64$_1$-412）、$-5.0 < a_1, a_2 < 5.0$ の範囲で設計変数を 0.2 刻みで変え、評価関数の分布を全て計算してみた。最適化問題の目的関数式(16)と制約条件式(17)を考慮し、評価関数として目的関数のみを扱う場合（式

(19)）と目的関数に制約条件をペナルティとして付加した場合（式(20)）の２つを考える。

$$F = C_\ell \tag{19}$$

$$F_{con} = C_\ell \cdot \exp(-100 \cdot |t/c - 0.15|) \tag{20}$$

図 11 には式(19)の分布、図 12 には式(20)の分布を示す。設計変数がともに負の場合、上下面が反対になって形状が物理的に正しく定義されないので、評価関数の値を強制的に零とした。

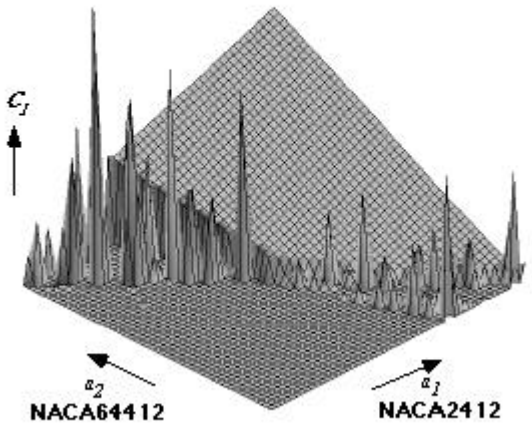

図 11　設計空間における揚力分布

図 11 において、2 つの基底翼型（NACA2412，NACA64$_1$-412）はともに最大翼厚比 12%なので、制約条件式(17)より

$$a_1 + a_2 = \frac{15}{12} \tag{21}$$

の関係を満たす直線上に求める最大値が存在すると予想される。しかし、2 つの基底翼型は異なる翼厚分布を持つため、評価関数の分布は単純な山にはならず、多くの鋭く切り立った山の分布になっている。さらに$a_1 = a_2 = 5.0$のとき最大揚力を示している。ところが、この場合最大翼厚比約 120%となり、事実上受け入れられない。これは非粘性流れを仮定していることから剥離がなく、キャンバーが大きくなることによって、計算上揚力が大きくなっているためである。また図 12 において、図 11 に比べると大幅に山の分布が減少していることが分かる。これは制約条件の影響の効果によるものであって、式(21)に沿った狭い範囲に山が分布している。

このような問題に勾配法を適用することを考えてみよう。まず、真の最適解を求めるには、最適値の十分近傍に初期値をとる必要がある。評価関数がこのように多くの鋭く切り立った山の分布を示す場合、これはほぼ不可能に近いであろう。というのは、最適化問題に勾配法を適用するとき、評価関数や制約条件が微分可能かつ凸関数である単峰性を仮定しているからである。実際、(1, 0)や(0, 1)のような自明な初期値では、図のような稜線にたどり着けな

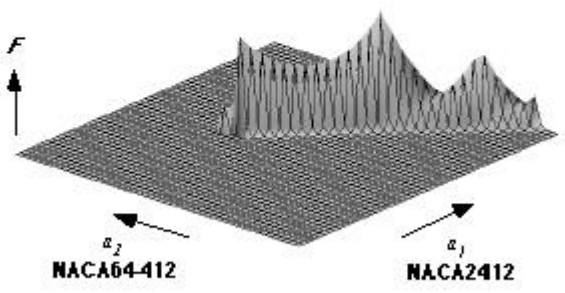

図 12　ペナルティつきの揚力分布

い。仮に、運良くこの稜線にたどり着いたとして、局所的な最適値を求めても、一番低い頂点のように現実的でない解に到達する可能性がある。

SA や GA は適用可能であるが、SA では初期値からこの稜線にたどり着くために、特に多くの繰り返し評価を必要とするであろう。また、2 つの手法ともに稜線を形作る特別な方向を認識するわけではないので、最適値を二次元的に探索するために、多くの無駄をしていると考えられる。また、このように目的関数の様子を調べてみると、(15)式による設計変数のとり方に再検討の必要があることがわかる。

2.5　4 設計変数による翼型形状最適化

前節で見たように、2 設計変数でさえも空力最適化問題は複雑であることが分かった。今度は、設計変数を 4 つに増やし、勾配法・SA・GA の 3 手法を用いて低速の高揚力翼型の最適化を行い、それぞれの最適翼型の性能を比較する。

図 13 にそれぞれの手法で最適化を試みたときの最適解への収束履歴を示す。収束履歴を比較するために、反復回数や世代数ではなく、パネル法による関数評価を行った回数を横軸に取った。以下、それぞれの手法の収束の様子を見てみよう。

勾配法の結果

勾配法には、Vanderpplats による最適化ソフト（ADS プログラム）[8]を用いた。ここでは ADS プログラムのオプションとして、勾配法の一種である実行可能方向法（Feasible Direction Method）を適用し、有限差分により評価関数の勾配を計算した。

4 つの設計変数のうち、一つだけを 1、残りを 0 とする 4 通りの初期値から最適化を行った。この 4 ケースは、最終的にそれぞれ異なる最適結果を示した。初期値 $(a_1,a_2,a_3,a_4)=(0,1,0,0)$ の場合が最も高い揚力となり、一方、$(a_1,a_2,a_3,a_4)=(0,0,1,0)$

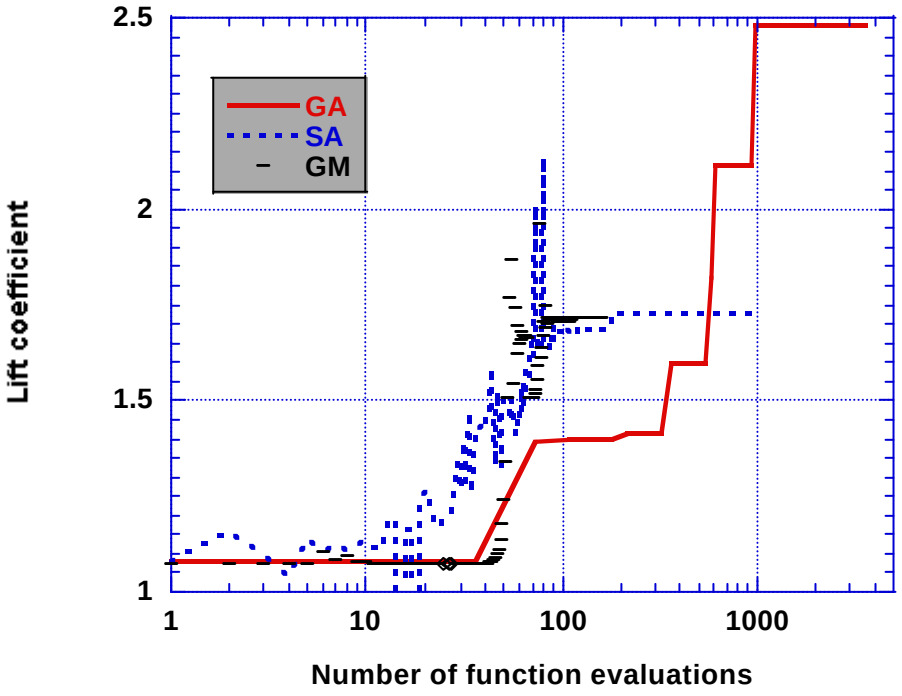

図 13　4 変数による最適化の収束履歴

の場合は、翼厚が負になったあと計算を続けることができなかった。図には、ベストの結果を載せてある。

SA の結果

ＳＡを用いる場合、クーリングスケジュールを調整する必要がある。今回は、試行錯誤を何回か繰り返してベストのパラメータを決めた。勾配法と同様に 4 通りの初期値から始めているにもかかわらず、全ての初期値で最適解が得られたので、SA は勾配法よりもロバストであると言える。今回最も高い揚力係数を示したのは初期値$(a_1, a_2, a_3, a_4) = (0,0,1,0)$の場合で、勾配法のときよりもわずかに性能は良くなった。勾配法と同様にベストの収束例を図に示した。

GA の結果

GA の各オペレータは、以下のように用いた。

- コード化：設計変数（基底翼型の係数部分）を実数型データのまま扱い、その 0.65（65%）に当たる大きさの部分と 0.35（35%）の大きさに当たる部分の２つにわけ、それらを順番に並べることによってコード化を行う（図 14）。本研究では設計変数は 4 つあるので、1 つの個体は 8 つの遺伝子により成立していることになる。
- 評価：評価関数には(20)式を用いる。
- 選択：選択手法には様々なものが考案されているが、本研究ではその中でも基本的なルーレット式選択法を採用する。この方法では、評価関数F_iを持つ個体iが親として選ばれる確率P_iが

$$P_i = \frac{F_i}{F_{sum}} \tag{22}$$

$$F_{sum} = \sum_{i=1}^{N} F_i \tag{23}$$

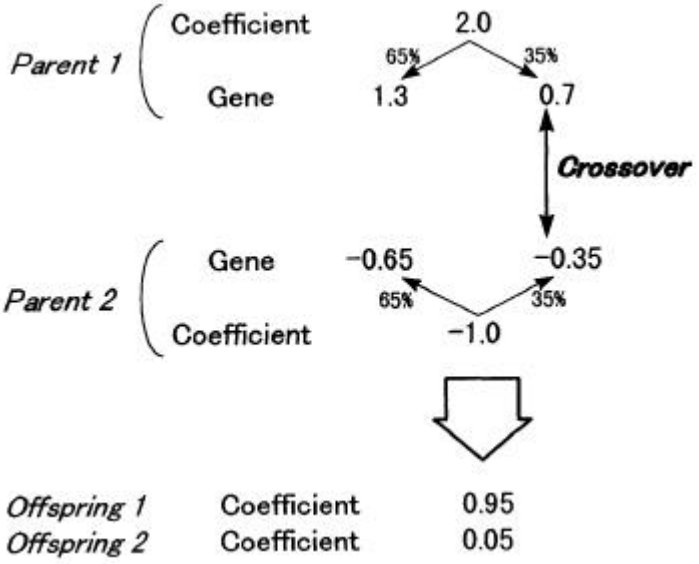

図 14　設計変数のコード化と交叉

で表される。ここで、N は個体数を示す。つまり、F_i が大きい個体ほど P_i も大きくなり、親として選ばれる頻度も増える。

- 交叉：設計変数の 0.35 の大きさに当たる部分を表す遺伝子を、選択された個体間ですべてを入れ換えている（図 14）。
- 突然変異：すべての遺伝子に対してその起こる可能性を調べ、もし発生した場合には *-0.1*～*+0.1* の間で乱数を発生させ、その値を突然変異の起こった遺伝子に加える。
- エリート戦略：現世代の中で最も評価の良い個体を 2 つ複製し、優先的にそのまま次世代に保存する。従って、残りの N-2 個体（N：個体数）を交叉や突然変異により形成する。

GA は勾配法や SA のようなある１つの初期値ではなく、ランダムに決められた初期集団から始まる。まず初めに 200 個体の集団を用いた。35 世代を越えたところで集団のほとんどが最適値に収束した。次に、この最適値を目標として、どこまで集団の個体数を減らせるかを試した。初めに 100 個体・50 個体を試したところ、同様の最適解を得た。そこで、40 個体・30 個体を試したところ、30 個体で最適解の揚力係数が減少した。図 13 には 36 個体を用い 27 世代で収束した例を示している。この結果から、GA は他の方法に比べて計算時間がかかるもののより大域的な最適解を探索するのに効果的であることが分かる。

一方、勾配法や SA ではより良い解を得るために、何通りかの初期値から計算をする必要がある。ところが、一般的に初期値を選択する方法がないので、ランダムに決めるしかない。それも今回のように 3 つや 4 つの初期値では話にならない。結果的に、勾配法と SA は効率という点でも不十分となり、全般的に見れば GA が 3 手法のなかで最も良い方法であると言える。

最適化を行う際に、GA は勾配に関する情報を必要としないため、目的関数や制約条件の微分可能性や、凸関数であるという条件が不要であり、初期の個体集団を大きくとれば大域的な最適解を探索することができる。しかしながら、GA は多点同時探索であるため、最適解が得られるまでに非常に多くの計算時間を費やしてしまうという欠点がある。例えば、３次元翼の流れ場計算に約 15 分程度時間を必要とすると、GA を適用して３次元翼の空力最適化問題を解く際、一個体の評価につき同様に約 15 分かかるといえる。従って、流体の最適化では全体の計算時間のうちほとんどが個体評価の部分に費やされているので、個体数 100、最大世代数 100 とした場合、単純に考えて最適解を得るまでに 15（分）×100（個体数）×100（世代数）=150000 分、つまり約 100 日も必要となってしまう。そこで一つの方法としては並列計算機を利用して各個体評価の計算を各プロセッサに分散させ、同時にすべての個体を評価することが考えられる。これによって計算時間の大幅な短縮になる。GA は並列化に適し、CFD はベクトル化に適していることから、GA と CFD による空力最適化は、ベクトル・パラレル型のスーパーコンピュータの性能をフルに活用することができるであろう。

なお GA の他、Genetic Programming (GP)、Evolution Strategy (ES)、Evolutionary Programming (EP)など同様のアイデアに基づく計算法があるが、それらはいわば流派の違いみたいなもので、現在は進化的アルゴリズム(Evolutionary Algorithms, EAs)として総称されるようになってきた。また進化的計算(Evolutionary Computation)とも呼ばれている。