第2回　　最適化ゼミ

指導 渡邉 (D1)　　佐野 (M1)
担当　チーフ：花田　サブチーフ：中村，下神納木

第 I 部
連続最適化問題に対するアルゴリズム

1　線形最適化問題

1.1　シンプレックス法

ここでは，問題を次の形に限定して話を進める．

$$\text{目的関数：} c^\top x \rightarrow \text{最小制約条件：} Ax = b, x \geqq 0$$

具体的な例を見てみる．以下のような目的関数と制約条件が与えられたとする．

$$\underline{\mathrm{Max}(-2x_1 - x_2)}$$
$$x_1 + 2x_2 \leqq 12$$
$$x_1 + x_2 \leqq 8$$
$$3x_1 + x_2 \leqq 18$$
$$x_1 \geq 0, x_2 \geq 0$$

このとき，目的関数，制約条件ともに限定された形にはなっていないので，形の修正が必要になる．具体的には，まず，目的関数については，

$\mathrm{Min}(-2x_1 - x_2)$ という風に，マイナスをかけることによって，最小化問題へと変換します．また，制約条件を型にはめるために，新たな変数を導入し，

$$x_1 + 2x_2 + x_3 = 14$$
$$x_1 + x_2 + x_4 = 8$$
$$3x_1 + x_2 + x_5 = 18$$

という形に変形することができる．ただし，$x_3 \geqq 0, x_4 \geqq 0, x_5 \geqq 0$ という条件がさらに付け加わることとなる．

基底解として，$x_1 = x_2 = 0, x_3 = 14, x_4 = 8, x_5 = 18$ とおいてやり，目的関数について

$$Z = Cx = -2x_1 - x_2$$

と置いてやり，係数比較をすると，x_1 を大きくしたほうが，全体としては小さくなるということがわかる．よって，x_1 を大きくするわけだが，どこまで大きくすることができるかということが問題となる．ここで，制約条件から

$$x_3 = 14 - x_1 - 2x_2$$
$$x_4 = 8 - x_1 - x_2$$
$$x_5 = 18 - 3x_1 - x_2$$

を導くことができ，$x_4, x_5, x_6 \geqq 0, x_2 = 0$ という条件があるので，それぞれから，

$$x_1 \leqq 14, x_1 \leqq 8, x_1 \leqq 6$$

という値を導くことができる．ここから，$x_1 \leqq 6$ が導かれて，$x_1 = 6$ とすると，

$$x_2 = 0, x_3 = 8, x_4, x5 = 0$$

を得ることができる．さらに，

$$x_1 = 6 - \frac{1}{3}x_2 - \frac{1}{3}x_5$$
$$Z = -12 - \frac{1}{3}x_2 + \frac{2}{3}x_5$$

ここでまた，基底変数は，

$$x_1 = 6 - \frac{1}{3}x_2 - \frac{1}{3}x_5$$
$$x_3 = 8 - \frac{5}{2}x_2 + \frac{1}{3}x_5$$
$$x_4 = 2 - \frac{2}{3}x_2 + \frac{1}{3}x_5$$

となることに注目し，次には基底変数として x_2 を入れることを考えるので，それぞれの式から，$x_2 \leqq 18, x_2 \leqq \frac{24}{5}, x_2 \leqq 3$ が得られる．これにより，$x_2 = 3$ として，$x_1 = 5, x_2 = 3, x_3 = 8, x_4 = 0, x_5 = 0$ となる．このとき，

$$Z = -13 + \frac{1}{2}x_4 + \frac{1}{2}x_5$$

となる．非基底変数の符号がともにプラスになったときが最適解であるという事が保証されており，ここで終了となる．

2　シンプレックスタブロー

では、この計算をシンプレックスタブローを用いて計算をしてみる．まず，Table 1 のような表を作る．初期状態

	-2	-1	0	0	0	0
x_3	1	2	1	0	0	14
x_4	1	1	0	1	0	8
x_5	3	1	0	0	1	18

Table 1　シンプレックスタブロー 初期段階

では、一番上の行には、目的関数における各変数の係数がきて、その下の部分は制約条件における変数の係数がくる．ここで，x_1 を基底変数に入れようとするのだが，それに対して基底変数のどれを非基底変数にするのかが問題になる．要するに，一番上の行の x_1 の列を 0 にして，下の行のどれかと入れ替えるのだが，どれと入れ替えるかということが問題となるのである．入れ替わる行を知るために，まず，制約条件の係数が現れているそれぞれの式において，一番右の列の値を，基底変数にしたい変数の係数（今回の場合は x_1 の係数なので，一番右の列の値）でそれぞれ割る．すると，上の行から順に，$14, 8, 6$ という値を得ることができる．そして，その中で一番小さい値を得ることのできた行と，一番上の行を入れ替えることになる．入れ替えを行う際には，まず，入れ替えられる行のそれぞれの要素を，基底変数の係数の値で割る．その結果，特に基底変数の係数の値は 1 となる．次に，それぞれの列の非基底変数から基底変数となる変数（今回は x_1）の係数の値が 0 となるように，その変数の係数を 1 にした行全体を何倍かしたものをそれぞれの列から引く．今回の場合は，一番上の行の x_1 の係数は -2 なので，-2 倍して引き，その他の行は x_1 の係数がそれぞれ 1 なのでそのまま引けばよい．そのような操作を行うと，表は Table 2 ようになる．このとき，一番上の行に注目して，そこにあるようその中に，負の要素があるかどうかを確認する．現時点では，x_2 の係数（2 列目）が，負の数として残っているので，作業を続行する．次に考えるのは，x_2 を基底変数としておき，どれかを非基底変数として，この二つを入れ替えることである．このときに，先ほどと同じように，2 行目以降の一番右の列の数を x_2 の係数（2 列目の値）で割り，それぞれで，$24/5, 3, 18$ という値を得ることができるので，その中の一番小さい数である，3 を得られた行に注目する．そこから先ほどと同じ作業を，今度はほかの行の x_2 の係数となる要素を 0 とする点に注意して実行し，Table 3 の表を得る．ここで一番上の行に注目すると，係数の要素（一番右の行を抜かした要素）がすべて正の値となっている．これが判定条件であり，この時点で終了となる．

	0	$-1/3$	0	0	$2/3$	12
x_3	0	$5/3$	1	0	$-1/3$	8
x_4	0	$2/3$	0	1	$-1/3$	2
x_1	1	$1/3$	0	0	$1/3$	6

Table 2　シンプレックスタブロー 第二段階

	0	0	0	$1/2$	$1/2$	13
x_3	0	0	1	$-5/2$	$2/3$	3
x_2	0	1	0	$3/2$	$-1/2$	3
x_1	1	0	0	$-1/2$	$1/2$	5

Table 3　シンプレックスタブロー 最終段階

$$x_1 = 5, x_2 = 3, x_3 = 3, x_4 = 0, x_5 = 0$$

となり，目的関数の最小値は，-13（表の右上角の値に -1 をかけたもの）になる．

3　非線形最適化問題

3.1　制約条件なしの最適化問題と降下法

非線形最適化問題において，関数の微分が重要な役割を果たす．最適化問題において，制約条件のない問題と制約条件のある問題とがあるが，制約条件のない非線形問題に対し，降下法と呼ばれる最適化手法がある．降下法とは，目的関数の値が減少するような方向にあるステップ幅だけ移動し続けることにより，最適解を求める手法である．降下法には，最急降下法や，ニュートン法がある．

降下法のアルゴリズム

(1) 初期点 x_0 を探索空間 E_n 内に選ぶ
(2) 探索方向 d_0 を定める（d_0 は降下方向に選ぶ）
(3)d_0 方向上のステップ幅を決め，次の点 x_1 を求める
　$x_1 = x_0 + t_0 d_0$（ただし，t_0 はステップ幅）

(2)〜(3) のプロセスを繰り返した結果，降下する割合が 0 に近づいたなら，その値を最適解とする．

降下法は，単峰な関数については，よほどのことがないかぎり[1]大域的最適解を見つけ出すことが可能であるが，複雑な多峰性の場合，局所解最適解に陥ってしまう問題がある．

3.1.1　最急降下法

2 回微分可能な関数 $f(x)$ に対して，$f(x)$ の偏変微分係数を要素とする n 次元ベクトルを点 fx における関数 $f(x)$ の勾配ベクトルとよび，以下の式で表す．

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_n} \\ \cdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

勾配ベクトルの方向はその点において関数が最も大きく増加する方向である．
最急降下法は，比較的アルゴリズムが簡単で，しかも，大域的収束性[2]が保証されている．最急降下法では，前節で記した降下法アルゴリズムにおける探索方向 d_0 に勾配ベクトルと逆方向のベクトル $-\nabla f(x)$ を利用する．したがって，

[1]解近傍で勾配が極端に大きな場合，ステップ幅が大きすぎると振動しつづける
[2]初期値をどこに選んだとしても，必ず局所解にたどり着くことができる

降下法アルゴリズムの (3) は $x_1 = x_0 - \nabla f(x)t_0$ となる．$\nabla f(x) \doteqdot 0$ になったなら，その値が最適解である．ステップ幅 t_0 については，一定値を用いる場合と，一定の割合でステップ幅を変化させる黄金分割と呼ばれる方法を用いる場合がある．ステップ幅が一定の場合，収束するまでに時間がかかり，場合によっては解近傍で振動することがある．また，初期値の取り方により，局所解に陥ってしまうことが多々ある．

3.1.2　ニュートン法

最急降下法は前節で述べたように，大域的収束性をもつという利点があるが，収束が遅くなる場合がしばしばある．ニュートン法は，$x_{n+1} - x_n$ の値を関数の勾配ベクトルをもとに変化させることによって収束の高速化を計ろうとする方法である．ニュートン法では，x_n から次の解 x_{n+1} を以下のように決定する．

$$x_{n+1} = x_n - \frac{\nabla f(x_n)}{\nabla^2 f(x)}$$

これを反復し，接平面 $\nabla^2 f(x)$ が水平になったなら，その解を最適解とする．この式は多変数の関数で適用可能である．多変数関数はイメージしにくく，式の意味の理解を得ることが難しいので，図示が可能な 1 変数関数で説明する．Fig. 1 は x_n から次の解 x_{n+1} を生成している様子である．

1 変数の場合，x_n における接線を求め，接線と横軸との交点を新しい解 x_{n+1} を生成している．関数の勾配が大きな

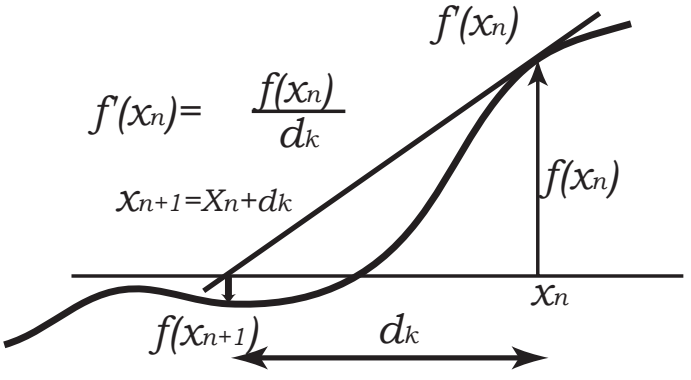

Fig. 1　ニュートン法

ときは，解に高速に接近するため，ニュートン法は最急降下法と比べものにならないほど速く解に収束する．ところが，ニュートン法も，最急降下法と同様に，初期値の取り方により，局所解に陥ってしまう．しかも，最急降下法が大域的収束性を持つのに対し，ニュートン法は局所的収束性[3]は持つが，大域的収束性を持たない．このことが，ニュートン法の実用面での大きな制約となっている．

ニュートン法の問題を解決する方法として，準ニュートン法というアルゴリズムがある．準ニュートン法は，収束する速度はニュートン法ほど早くはないが，大域的収束性を持つ．準ニュートン法は一般的に用いられている．

3.2　制約なし最適化問題と 1 次元探索

最も簡単な最小化問題とは，目的関数が 1 変数関数の最小値を求める問題である．このような問題の解を求めることを 1 次元探索という．この問題に対して，先ほど説明した最急降下法，ニュートン法を用いても構わないが，ここでは 1 変数関数に限ってのみ用いることができるアルゴリズムである黄金分割を用いた探索法がある．ただし，この探索法をもちいる前に，まず，最小値が存在する区間を求めなければならない．そして，区間が定まったなら，このアルゴリズムで，区間を絞っていく．

3.3　黄金分割による探索法

黄金分割による探索法は黄金比 $\tau = \dfrac{1+\sqrt{5}}{2}$ を用いた探索法である．

【黄金分割による探索法のアルゴリズム】

[3]初期値を解の十分近くに選んだなら，その解への収束が保証される

(1) 最小点が存在する区間 (x_1, x_2) に $x_1 < x_3 < x_4 < x_2$ となるような 2 点 x_3, x_4 を次のように選ぶ.

$x_3 = x_1 + \dfrac{\tau - 1}{\tau}(x_2 - x_1)$

$x_4 = x_1 + \frac{1}{\tau}(x_2 - x_1)$

(2) 以下の判定条件により，x_1，x_2 のどちらかを新たな点 x_3，x_4 と入れ替える.

- $f(x_3) < f(x_4)$ ならば $x_1 = x_3$ とし，
- $f(x_3) < f(x_4)$ ならば $x_2 = x_4$ とする.

(1)，(2) を繰り返し，あらかじめ定めた十分小さな正の値 ϵ に対して $(x_2 - x_1) < \epsilon$ が成り立てば，その時点で打ち切り，区間 $[x_1, x_2]$ を解が存在する区間とする.

3.4 制約条件つきの最適化問題

制約条件がない問題においては，点 x^* 局所的最適解であれば目的関数の勾配が 0 になる．けれども，制約つき問題においては，その局所的最適解が Fig. 2 のように実行可能領域の境界上に存在することが多く，その点で目的関数の勾配が 0 になるとは限らない.

制約つき問題に対しては，目的関数だけでなく制約条件に含まれる関数も考慮する必要があるののので，使用するアル

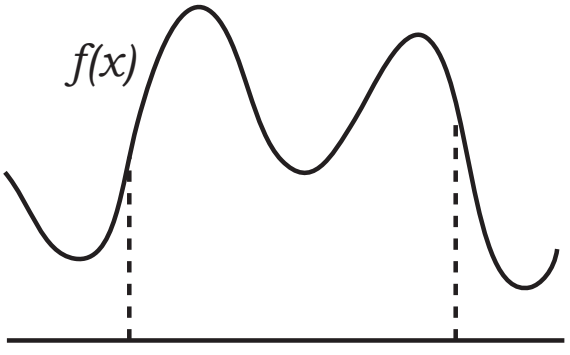

Fig. 2　境界上の最適解

ゴリズムは制約なし問題と異なる．制約条件つき問題に対する手法に，ペナルティ法などがある．ペナルティ法では，ペナルティ関数という関数を用いることにより，制約つき最適化問題を制約なし最適化問題に変換し，降下法などで解を求める方法である．ここではアルゴリズムの説明を省くが，これらのアルゴリズムを用いて最終的に求まった解が局所最適解であるか判定するキューン・タッカー (Kuhn・Tucker) 条件を紹介する.

3.4.1 キューン・タッカー条件

【キューンタッカー条件】

x^* が制約つき問題の局所最適解であり，制約条件 c_m のうち，活性な制約条件[4] $c_i (i = 1, 2, \cdots l, 0 \leq l \leq m)$ の勾配ベクトル $\nabla f(x^*)$ と $\nabla c_i(x^*)$ が，その点において 1 次独立ならば，式 (1) が成り立つようなベクトル $u^* = (u_1^*, u_2^*, \cdots, u_m^*)$ が存在することがいえる.

$$
\begin{cases}
\nabla f(x^*) + \sum_{i=1}^{m} u_i^* \nabla c_i(x^*) = 0 & \\
c_i(x^*) = 0 & (i = 1, 2, \cdots, l) \\
c_i(x^*) = 0,\ u_i^* \geq 0 & (i = l+1, 2, \cdots, m) \\
c_i(x^*) < 0,\ \Rightarrow \quad u_i^* = 0 & (i = l+1, 2, \cdots, m)
\end{cases}
\tag{1}
$$

この条件をキューン・タッカー条件という．式 (1) は，活性な制約条件 $c_i (i = 1, 2, \cdots l, 0 \leq l \leq m)$ の解 x^* における勾配ベクトル $\nabla c_i(x^*)$ が 1 次独立の場合，その点における目的関数 $f(x)$ の勾配ベクトル $\nabla f(x^*)$ と $\nabla c_i(x^*)$ の和の

[4]最適解において，等式が成り立つ制約条件

ベクトルをスカラー倍したベクトルが釣り合うことを意味している．式だけでは分かりにくいので，例を用いて説明する．
以下のような目的関数，制約条件を考えます．

$$
\begin{aligned}
&\text{目的関数：} f(x) = (x_1 - 1)^2 + (x_2 - 2)^2 \quad \rightarrow \quad \text{最小} \\
&\text{制約関数：} c_1(x) = {x_1}^2 + {x_2}^2 - 2 \leq 0 \\
&\qquad\qquad c_2(x) = -x_1 + x_2 \leq 0 \\
&\qquad\qquad c_3(x) = -x_2 \leq 0
\end{aligned}
\tag{2}
$$

この問題の最適解は目的関数の等高線と実行可能領域が交わった点 $x^* = (1,1)^\top$ である．(Fig. 3 参照) x^* における目的関数の勾配ベクトルと活性な制約条件 $\nabla f(x^*), \nabla c_1(x^*), \nabla c_2(x^*)$ が釣り合っているような形になっている．このことは，式 (3) を満たすような $u_1^* \geq 0$,$u_2^* \geq 0$ が存在することを示します．

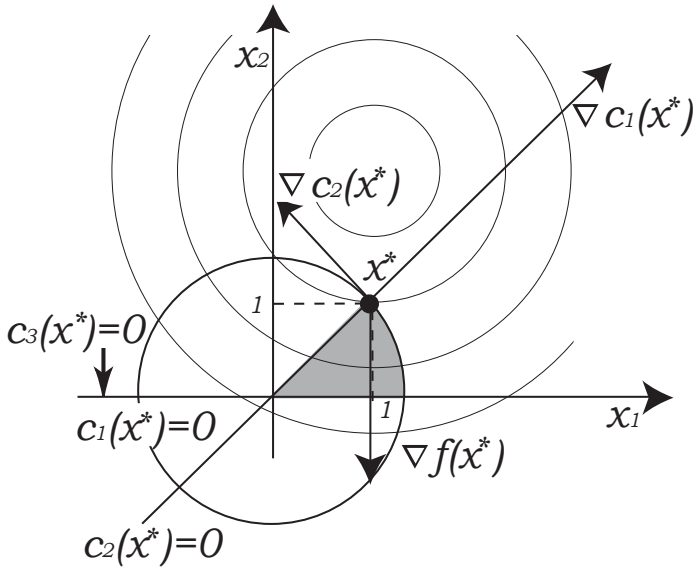

Fig. 3　目的関数と制約関数の勾配

$$
\nabla f(x^*) + u_1^* \nabla c_1(x^*) + u_2^* \nabla c_2(x^*) = 0 \tag{3}
$$

式 (3) は，活性な制約条件のみを含んだ式となっているが，次のように書き換えることができる．

$$
\begin{cases}
\nabla f(x^*) + u_1^* \nabla c_1(x^*) + u_2^* \nabla c_2(x^*) + u_3^* \nabla c_3(x^*) = 0 \\
c_i(x^*) = 0 \quad (i = 1,2,3) \\
c_i(x^*) = 0,\ u_i^* \geq 0 \quad (i = 1,2,3) \\
c_i(x^*) < 0,\ \Rightarrow \quad u_i^* = 0 \quad (i = 1,2,3)
\end{cases}
\tag{4}
$$

式 (4) の第 1 式は見かけ上すべての制約関数を含んでいるが，第 4 式より，活性な制約ではない 3 番目の制約条件の係数 u_3^* は 0 になるので，式 (3) と式 (4) は実質的に同じものを表している．そして，式 (4) を満たす局所解 $x^* = (1,1)^\top$ は式 (1) を満たしいる．
局所解はキューン・タッカー条件を満たすが，最終的に見つかった解がキューンタッカー条件を満たすからといって，その解が局所解であるかどうかは分からない．ただし局所解は必ずキューンタッカー条件を満たすので，最終的に求まった解が条件を満たしていない場合は確実に局所解ではないことが分かる．この条件により，解候補をある程度挙げることができる．

第 II 部
DOTの使い方

4 DOT とは？

4.1 DOT の特徴

最適化ソフトウェア DOT(Design Optimization Tools) は，最適化問題を解析するための汎用最適化パッケージである．DOT による最適化解析では，最適化問題の設定と最適化評価を行うメインプログラムを作成する．この中から DOT 及び必要に応じ解析ソルバーの呼び出しをすることになる．最適化問題の設定では，設計変数の初期値，上限下限値，応答に関する制約条件，さらに目的関数の評価と最小・最大化を定義し，DOT ルーチンへの引数パラメータとして与え，DOT の中で最適化探査が実行される．その際には，最適化プログラムの中では，設計変数を X ベクトル，制約条件を G ベクトルで定義している．一方対象とする最適化問題は単純な線形関数から，複雑な陰関数まで，その内容や規模に制限はない．そして，最適化理論に関する特別な知識は必要ない．単純ではあるが，必要に応じて最適化パラメータを調整することで多様な最適化が可能となる．そして，非線形，線形を問わず良好な解を高速に得ることができる．

4.2 最適化ソフト DOT の使用アルゴリズム

Table 4　DOT による使用アルゴリズム

最適化方法	修正可能方向法, 逐次線形計画法, 逐次 2 次計画法
制約条件の無い場合	共役傾斜法 (FR 法、BFGS 法)
最適化パラメータの自動設定	必要に応じ再定義可能

以上の Table 4 のアルゴリズムを使用して,DOT では問題を解いている. そして, 実際には以下の手順によって問題を解く.

1. 設計変数に摂動を与え, 目的関数と臨界制約条件についての感度計算
2. 制約条件を満足し, 目的関数を改良する探査方向を求める
3. 最適化探査方向での最適解を求める
4. 最適化の収束判定 (収束、解析続行)

5 DOT プログラム

5.1 DOT のソース構成とコンパイル

DOT のソースは，計算精度の違うタイプとして，float 型と double 型の二種類存在する．前者が DOT と呼ばれており，これは main.c，dot1.c，dot2.c，dot3.c，dot4.c，dot5.c，dot6.c のソースで構成されている．そして，後者が DDOT と呼ばれており，これは main.c，ddot1.c，ddot2.c，ddot3.c，ddot4.c，ddot5.c，ddot6.c のソースで構成されている．メインルーチンは，main.c 中に存在している MAIN_() に書かれており，ユーザが変更する部分は基本的に，この中の変数値（初期設定，各設計変数の制約条件など）と目的関数（制約条件）だけである．

また，変更部分をファイルからの入力に改良したバージョンがある．それは，DOT に対しては，main2.c，o_file.c，dot1.c，dot2.c，dot3.c，dot4.c，dot5.c，dot6.c + input.txt であり，DDOT に対しては，o_file.c，ddot1.c，ddot2.c，ddot3.c，ddot4.c，ddot5.c，ddot6.c + input.txt である．各プログラム用に makefile が書かれており，基本的にはそれをコンパイルすれば実行可能である．しかし，コンパイルを行う際には，DOT の中で使用されている関係上,,「f2c.h」が使用できる環境であること必要である．この「f2c」パッケージは，コンパイル時に Fortran(g77) プログラムを c(gcc) プログラムへ自動的に変換して，プログラマーの負担を減らすフロントエンドプログラムである．

コンパイルが成功すると，dot.exe（DOT の場合），又は ddot.exe（DDOT の場合）という名前の実行ファイルが作成される．

5.2 DOT プログラムのパラメータ

- method
  最適化手法を指定する変数を表す．DOT，DDOT では 3 種類の非線形制約ありの最適化手法が用意されており，それぞれ 1〜3 の番号を指定することにより手法を決定することが出来る．通常は「1」のまま使用する．

- ndv
  設計変数の数を表す．つまり，2 変数の場合には ndv=2 とする．

- ncon
  制約条件の数を表す．ただし，DOT,DDOT では各変数ごとの制約条件はこの場合含まれない．つまり 0 ≦ x1 ≦ 1 といった制約条件はこの場合の制約条件数に含まれない．勿論，2x1+3x2¿0 といった制約条件は含まれる．

- iprint
  DOT,DDOT では，出力形式が 3 つのレベルに分類されていて，それぞれの出力形式を指定することが出来る．レベルは 1〜3 の 3 段階に分かれておりレベルが上がるほどより詳細なデータが出力されるようになっている．解のみを知りたいような場合にはレベル 1(iprint=1) で十分である．

- x,xl,xu
  x は各設計変数の値，xl は設計変数の下限制約，xu は設計変数の上限制約を表す．上限制約，下限制約ともに各変数別に設定することが可能である．注意点として各変数の初期点は必ず制約条件内に存在するように設定しなければいけない．プログラムが期待通り動作しない恐れがある．

- obj
  目的関数の値を表す．関数 eval_() により具体的な目的関数計算が行われている．

- g
  制約条件の値を表す．obj 同様，関数 eval_() により具体的な計算が行われている．注意点として，この g[] の値は負の時が真，つまり制約条件内であり，正の時には制約条件外であると見なす．例えば，式の変換としては，以下のように設定する．

$$x_2 \geq -0.5x_1 + 3 \tag{5}$$

  (1) ならば, 以下の (2) のように変換し,

$$0 \geq -x_2 - 0.5x_1 + 3 \tag{6}$$

  (2) の形にしたところで,

$$g[0] = -x[1] - 0.5x[0] + 3 \tag{7}$$

  (3) とする．
  x[1] や x[0] は各設計変数に対応する．この式の対応からもわかるように，g[] は値が負となる時，制約条件を守っていると判断する仕組みになっている．

原本における変数設定は，main.c の MAIN_() に直接書き込むことにより行う．また，関数，制約条件の設定も同様に main.c の eval_() を直接書き換えることにより行う．特に，制約条件の記述時には，必ず記述式が負の時に真となるように気を付けるようにする．

5.3 注意点

DOT プログラムのデフォルトでは，設計変数と制約条件の上限は，それぞれ 100 と 50 に設定されている．もし，この設定を越えるような関数最適化を行う場合には，main.c の MAIN_() における配列 g(制約条件)，x(設計変数)，xl(設計変数の下限制約)，そして xu(設計変数の上限) を宣言時に，必要な配列数に変更しなければいけない．ただし，あまりにも多くの設計変数，制約条件がある時，他の配列変数にも影響を及ぼす場合があるので，その場合には，rprm, wk などの変数配列も変更する必要がある．

5.4 実行結果の確認

実行結果データの見方について，簡単に説明します．以下に示すデータ以外にも，有用なデータが結果として表示される．これらの表示結果については，パラメータである「iprint」に依存する．以下のデータは，最適解のみを知りたい場合に必要なデータである．

- OBJECTIVE : 最適解
- DECISION VARIABLES : 得られた最適解の設計変数の各値
- CONSTRAINTS : 制約条件

6 最適化ソフト DOT の使用法

最適化ソフトウェア DOT を使用するにあたって，まず必要なファイルのインストールについて述べる．その後，例題を用いて，DOT の使用法について述べる．

6.1 必要なファイル

最適化ソフトウェア DOT を使用するために必要なファイルを以下に示すので，共有フォルダ（`/Shimo/share`）から各自コピーしておくようにする．

- フォルダ optimization
- フォルダ Cgwin（Windows 上で最適化ソフト DOT を使用する場合）
- フォルダ f2c（Windows 上で最適化ソフト DOT を使用する場合）

6.2 使用方法

以下に，Linux 上での実行方法と Windows 上での実行方法を示す．

- Linux 上での実行方法

1. コピーしたフォルダ optimization を Forte クラスタにアップする．
2.Forte クラスタに login する．
3. フォルダ w_dot，又はフォルダ w_ddot に移動する．
4. コマンド make clean を実行する．
5. コマンド make を実行する．(このコマンドで実行ファイルが完成する．)
6. コマンド./*****(作成されたファイル dot，又は ddot) を実行する．

- Windows 上での実行方法[5]

1. 共有フォルダ（`/Shimo/share`）のファイル setup.exe を実行して，Cygwin というツールを各自インストールする．
2.Cygwin/bin にパスを通す．
3. 共有フォルダからコピーしたフォルダ f2c の中の f2c.exe をコピーして，Cygwin/bin へ置く．
4. フォルダ f2c の中にあるライブラリである libf2c.a を，Cygwin/lib に置く．
5. フォルダ f2c の中にあるヘッダ f2c.h を，Cygwin/lib/gcc-lib/i686-pc-Cygwin/2.95.3-2/include に置く．
以上で，最適化ソフト DOT が使用できる環境になりました．
6. コマンドプロンプトで，コマンド bash を実行する．
7. フォルダ w_dot，又はフォルダ w_ddot に移動する．
8. コマンド make clean を実行する．
9. コマンド make を実行する．(このコマンドで実行ファイルが完成する．)
10. コマンド./[作成されたファイル] を実行する．

[5]長谷氏の協力により実現

6.3 例題

以下の問題を最適化ソフト DOT と GA の両者で解いて，比較してみる．

x_1 と x_2 に対して,${x_1}^2 + (x_1 + x_2)^2 + {x_2}^2$ を最小にすることを目的とする. その上での制約条件は,

$$-5.12 \leq x_1 \leq 5.12 \tag{8}$$

$$-5.12 \leq x_2 \leq 5.12 \tag{9}$$

とする.

最適解は,(R_1, R_2) として,$(0, 0)$ である.

この問題の目的関数を以下の Fig. 4 に示す.
両者で実行した結果を以下の Table 5 に示す.

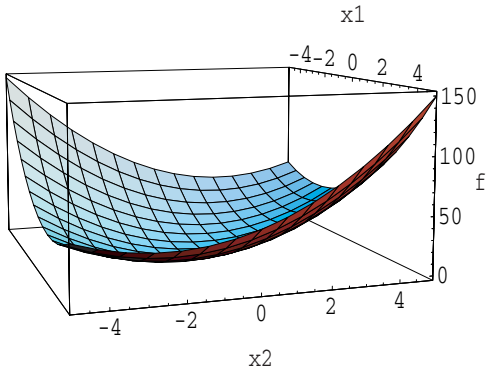

Fig. 4　目的関数

Table 5　DOT と GA による実行結果

手法	実行結果	実行時間
DOT	0.00000E+00	0m0.050s
GA	0.00000E+00	0m0.130s

結果より，簡単な目的関数，制約条件では，最適化ソフト DOT が優れていることがわかる．今回の GA の実行では，世代数 64，評価回数 1228 という計算量であった.

参考程度に今回使用した GA のパラメータを Table 6 に示す.

Table 6　GA のパラメータ

個体数	20
エリート	1
染色体長	20
設計変数	2
設計変数の下限	-5.12
設計変数の上限	5.12
選択	ルーレット選択
交叉 (率)	1 点交叉 (0.8)
突然変異率	0.05