第 3 回 最適化ゼミ

ゼミ担当者　：　勝崎　俊樹，及川　雅隆，谷口　義樹
指導院生　　：　小野　景子，花田　良子
開催日　　：　2002 年 5 月 18 日

1　線形最適化問題

1.1　シンプレックス法

前回のゼミで，線形最適化問題の解法の一例としてシンプレックス法を挙げ，グラフで解く方法について示した．線形最適化問題では，最適解は制約条件からなる領域（凸多角形）のどこかの頂点にある．シンプレックス法は，それらの頂点を一つ一つ調べて，解を見つける方法である．今回，以下のような例を用意した．

＜簡単なシンプレックス法の例＞

ある会社が，製品 A，B を売ろうとした．それらを製作するための原料を石油，鉄，パルプとする．それぞれ１個を製作するのに必要な材料の量を以下の表の通りとする．

Table 1　材料と製品との関係

	石油	鉄	パルプ	単価
A	1kg	1kg	3kg	2 万円
B	2kg	1kg	1kg	1 万円
材料の量	14kg	8kg	18kg	

このとき，手持ちの材料の範囲内で売上高を最大にするためには，A，B をそれぞれいくつ作ったらよいのかを考える．今回は，次章のシンプレックスタブローを用いた説明が分かりやすくなるように，その流れに沿った解法を行ってみる．

a) **目的関数の設定**

売上高は１つにつき，A が 2 万円，B が 1 万円なので，A を x_1 個，B を x_2 個売ったとすると，売上の合計は $2x_1 + x_2$ となる．今回の目的はこの合計額をできる限り大きくすることなので，$Max(2x_1 + x_2)$ が目的関数となる．

b) **制約条件の設定**

石油は A を作るのに 1kg，B を作るのに 2kg 必要だが，その合計量は 14kg しかないので，

$$x_1 + 2x_2 \leq 14$$

となる. 同様に，鉄，パルプについても考えると，

$$x_1 + x_2 \leq 8$$
$$3x_1 + x_2 \leq 18$$

となる．

c) **非負条件の設定**

今回のように,「ものの個数」などを扱う最適化問題では，数が負になることはありえない．そのことから，以下のような条件が与えられる．

$$x_1 \geq 0, x_2 \geq 0$$

d) **シンプレックス法による解法**

ここまでで得られた条件をまとめると，以下のようになる．

$$Max(2x_1 + x_2)$$
$$x_1 + 2x_2 \leq 14$$
$$x_1 + x_2 \leq 8$$
$$3x_1 + x_2 \leq 18$$
$$x_1 \geq 0, x_2 \geq 0$$

これをシンプレックス法の標準形に直すことで解を求める．

・目的関数の変形

今回の目的関数 $Max(2x_1 + x_2)$ は，条件を負にしてやることで，最小化問題に書き換えることができる．今回のシンプレックス法では，この手法を用いて計算を行う．最小化問題に置き換えられた目的関数は以下のようになる．

$$Min(-2x_1 - x_2)$$

・制約条件の変形

次に，制約条件をシンプレックス法の標準形に当てはめるために，次のように変形させる．シンプレックス法の標準形とは，それぞれの制約条件に条件を満たす変数

を加えることで，不等式を等式として表すものである．今回の条件では，以下のようになる．

$$x_1 + 2x_2 + x_3 = 14 \quad (1)$$

$$x_1 + x_2 + x_4 = 8 \quad (2)$$

$$3x_1 + x_2 + x_5 = 18 \quad (3)$$

このとき，不等号の関係から $x_3 \geq 0$，$x_4 \geq 0$，$x_5 \geq 0$ となっている．

また，目的関数を見ると，$Min(-2x_1 - x_2)$ であるので，係数比較より x_1 が大きい方が全体として小さくなっていくことが分かる．あとは，いかにこの条件を満たす x_1 の最大値を探すかが重要となってくる．

ここで，非負条件を満たしつつ最小となる x_2 である，$x_2 = 0$ を，式 (1)〜式 (3) に代入する．それを変形すると，以下のようになる．

$$x_3 = 14 - x_1 \quad (4)$$

$$x_4 = 8 - x_1 \quad (5)$$

$$x_5 = 18 - 3x_1 \quad (6)$$

ここで，$x_3 \geq 0$，$x_4 \geq 0$，$x_5 \geq 0$ の条件より，式 (4)〜式 (6) よりそれぞれ

$$x_1 \leq 14, x_1 \leq 8, x_1 \leq 6$$

という値を導くことができるので，$x_1 \leq 6$ を導くことができる．

ここで，もっとよい解はないかを確認する．$x_1 = 6$ を導くのに利用した (3) を変形して，

$$x_1 = 6 - \frac{1}{3}x_2 - \frac{1}{3}x_5 \quad (7)$$

$$x_3 = 8 - \frac{5}{3}x_2 + \frac{1}{3}x_5 \quad (8)$$

$$x_4 = 2 - \frac{2}{3}x_2 + \frac{1}{3}x_5 \quad (9)$$

今回の式の変形の目的は，より大きい x_2 を求めることにある．そのため，式 (7)〜式 (9) を式 (10)〜式 (12) のように変形する．

$$x_2 = 18 - 3x_1 - x_5 \quad (10)$$

$$x_2 = \frac{24}{5} - \frac{3}{5}x_3 + \frac{1}{5}x_5 \quad (11)$$

$$x_2 = 3 - \frac{3}{2}x_4 + \frac{1}{2}x_5 \quad (12)$$

x_2 の範囲は式 (10)〜式 (12) から，それぞれ，$x_2 \leq 18$, $x_2 \leq \frac{24}{5}$, $x_2 \leq 3$ が得られる．すべての条件を満たすものは $x_2 \leq 3$ なので，その最大値 $x_2 = 3$ を選ぶ．すると，式 (7)〜式 (9) より $x_1 = 5, x_2 = 3$ となる．このとき，目的関数は $min(-13)$ となる．この目的関数は初めに負をかけることにより，最大化問題を変形したものなので，もう一度正条件に戻すと，この問題の真の目的関数 $max(2x_1 + x_2)$ は $x_1 = 5, x_2 = 3$ のとき満たされるといえる．よって解は，

”$x_1 = 5, x_2 = 3$ のとき，最適解 13 を得る ”

ということになる．

1.2 シンプレックスタブロー

シンプレックスタブローとは，別名シンプレックス表とも呼ばれ，前述のシンプレックス法をより簡単に，より効率的に行うために利用されるものである．まず，シンプレックスタブローを利用するために，Table2 の表を製作する．この表において，z は目的関数を，その他の行は制約関数を，そして列はそれぞれの要素の値を示す．

Table 2　シンプレックスタブローの初期状態

基底変数	x_1	x_2	x_3	x_4	x_5	定数項
z	-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

Fig2 は最小化問題に変換した目的関数と式 (1)〜式 (3) に示された初期の制約条件を忠実に示しており，最適解に近づけながら更新していく．そのアルゴリズムは，Fig. 1 のようになる．

Fig1 に示したようなアルゴリズムを用いて，実際に Table2 を変形していく．まず，

目的関数の行にマイナスの値がないかを調べ，その中で絶対値の大きいものを選ぶ．

それぞれの列で　で選択された行の数字を用いて定数項を割る．このとき，その商が最も小さい列を選択し，その行の基底変数と　で選択された変数を入れ替える．

Table 3　シンプレックスタブローの第 2 段階

基底変数	x_1	x_2	x_3	x_4	x_5	定数項
z	-2	-1	0	0	0	0
x_3	1	2	1	0	0	14
x_4	1	1	0	1	0	8
x_1	3	1	0	0	1	18

で選択された列において，　で選択された行との交差点（ピボット）のみ 1 に，その他の数字が 0 になるように演算する．

Table 4　シンプレックスタブローの第 3 段階

基底変数	x_1	x_2	x_3	x_4	x_5	定数項
z	0	-1/3	0	0	2/3	12
x_3	0	5/3	1	0	-1/3	(14-6)=8
x_4	0	2/3	0	1	-1/3	(8-6)=2
x_1	1	1/3	0	0	1/3	6

目的関数の行にマイナスの値が残っていれば，同様の動作を繰り返す．

Table 5　シンプレックスタブローの第 4 段階

	x_1	x_2	x_3	x_4	x_5	定数項
z	0	0	0	1/2	5/6	13
x_3	0	0	1	-9/10	-1/30	31/5
x_2	0	1	0	3/2	-1/2	3
x_1	1	0	0	-1/2	4/9	5

目的関数の行にマイナスの値がなくなったので終了．

よって，解として，$x_1 = 5, x_2 = 3$，そして最適解-13 を得ることができた．実際の問題では，目的関数に負をかけたので，さらに (-1) をかけてやれば，正しい最適解 13 を得られる．ただし，この方法で全ての線形最適化問題が解けるとは限らない．そこでこの方法を応用したものに，2 段階シンプレックス法というものも存在する．2 段階シンプレックス法では，相対費用係数という概念をシンプレックス法に導入することで，2 回シンプレックス法を行う．

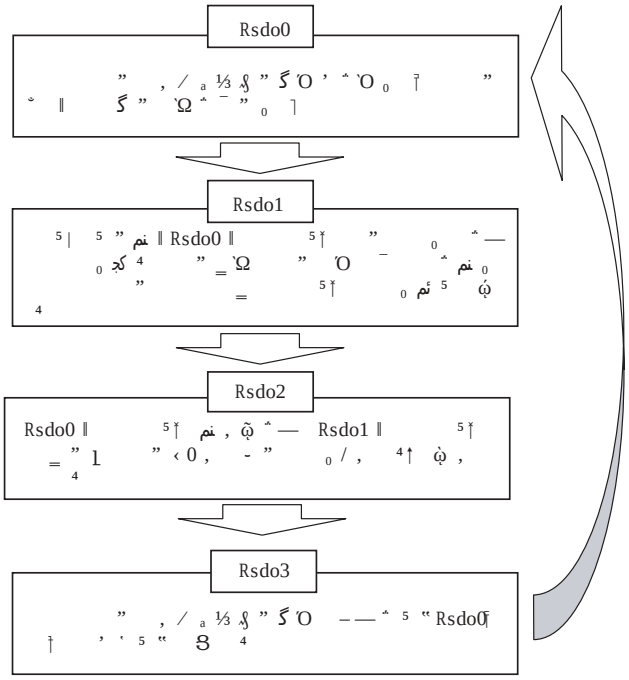

Fig. 1　シンプレックスタブローのアルゴリズム

2　離散最適化問題の概要

2.1　組合せ問題とは

離散最適化問題は，目的関数や，制約条件が離散的な点の集合である最適化問題のことである．離散的な問題に，組合せ最適化問題というものがある．

組合せ最適化問題とは，いろいろなものの組合せの中で，与えられた制約条件を満たし，かつ，目的関数を最小化するものをみつける問題でである．組合せ最適化問題には TSP[1]，スケジューリング問題に代表されるネットワーク上の最適化問題，およびナップザック問題のような組合せ最適化問題の解の効率的列挙などがある．ここでは，TSP，スケジューリング問題，ナップザック問題について紹介する．

2.2　TSP

TSP とは，Fig. 2 のように，都市数とそれぞれの都市間の道のりが分かっているとき，セールスマンが N 個ある都市を必ず 1 度だけ通って巡回したとき，総経路長を最短にするような都市の巡回経路を見つける問題である．都市数が N 個のとき，考えられる巡回経路の種類は，N 個の要素の最適な並び替え（順列）であるので (N-1) ! /2 個である．例えば，N=10 の場合，約 18 万だが，N=20 となると約 6×10^{16} となってしまう．このことから分かるように，N が大きくなると，この組合せは爆発的に多くなる．このように現象を組合せ爆発という．この場合もすべての巡回経路を調べることは実際に

[1] Traveling Salesman Probrem，巡回セールスマン問題

は不可能で，都市の配置に法則性がなければ厳密解を求めることはできない．

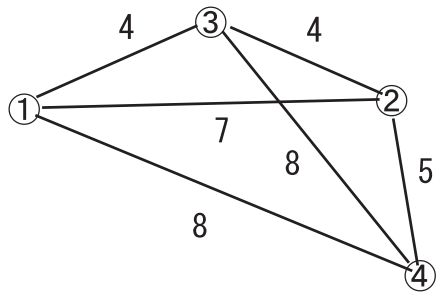

Fig. 2　TSP

2.3　スケジューリング問題

スケジューリングとは, いくつかの, やらなければならない仕事があるときに, どういう順番で仕事を進めればもっとも速いか, あるいは都合が良いか, その最適な仕事をやる順番を求める問題のことをいう. ここでは，スケジューリング問題の例として生産スケジューリング問題，ジョブショップスケジューリング問題，プロジェクトスケジューリング問題，ナーススケジューリング問題を説明する．

2.3.1　生産スケジューリング問題

生産活動を円滑に行うためには，製品を作る行程での作業の所要時間や順序関係を考慮しなければならない．その実際の生産加工にあたって，どの生産設備（機械）で，いつ加工するかという，個々の要素作業のための具体的で詳細な時間日程が必要となる．その最適ないし実行可能な詳細実施プログラム（スケジュール）を決定することを「生産スケジューリング」と呼ぶ．制約条件により，様々な問題に分けられるが，この中にジョブショップスケジューリング問題がある．

2.3.2　ジョブショップスケジューリング問題

ジョブショップスケジューリング問題とは，Fig. 3で表されるように複数の作業の直列な接続によって接続されており，資源である機械を用いて製品（ジョブ）を加工する順序を，いくつかの評価基準に基づいて決定する問題である．このとき，資源を同時に使用することはできないものとされる．

2.3.3　プロジェクトスケジューリング問題

プロジェクトスケジューリング問題は，ジョブショップスケジューリング問題をさらに拡張した問題で，工場におけるスケジューリング問題のみではなく，建築工事や製品開発などより広い適用範囲をあげることができる．この問題では，行程に属する作業が直列である必要がなく，分岐や合流があってもかまわない．また，利用

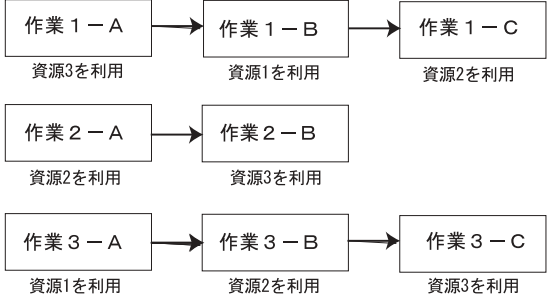

Fig. 3　ジョブショップ・スケジューリング問題

する資源は同時に複数の作業で共有することが可能であるものとする．プロジェクトスケジューリング問題は，PERT（Problem Evaluaton and Review Technique）によって一般的に扱われる．

2.3.4　ナーススケジューリング問題

ナーススケジューリング問題とは，昼夜３交代制で２０名程度の看護婦達の一ヶ月勤務表をいくつかの制約条件のもとで，各個人の不満が最小限になるように作成する問題である．(Fig. 4 参照)

Fig. 4　ナーススケジュール問題

2.4　ナップザック問題

ナップザック問題は，複数の物体（個々の物体は異なる重さと価値を持つ）が与えられた時に，重さがある範囲以内でいくつかの物体を選択し，その時の価値の合計が最大になるような選択の方法を見つけるという問題である．さらに，選ぶべき各要素をたかだか１回しか選べない場合は，特に 0-1 ナップザック問題と呼ばれている．

3　DOT の使い方

3.1　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，線形，非線形を問わず良好な解を高速に得ることができる．

3.2 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．

```
/****************
float により値を返す非線形制約ありの
最適化を行うプログラム dot の main 文
今回は例として関数-2x+y に関する最適化を行っている．
************/

#include "f2c.h"
#include "math.h"

/* SAMPLE PROGRAM.  EQUALITY CONSTRAINTS. */

MAIN__()
{
  /* System generated locals */
      integer i__1;
  /* Builtin functions */
  /* Subroutine */ int s_stop();

  /* Local variables */
  extern /* Subroutine */ int eval_();
  static integer info, ncon, iprm[20];
  static real rprm[20];
  static integer nrwk;
    static real g[50];/*制約条件の MAX を 50 に設定*/
  static integer i__;
    static real x[100];/*設計変数の数を MAX100 に設定*/
    static integer nriwk;
    static real wk[800],xl[100],xu[100];
  static integer method, minmax, iprint;
  static real obj;
  extern /* Subroutine */ int dot_();
  static integer ndv, iwk[200];

/*     DEFINE NRWK, NRIWK. */
  nrwk = 800;
  nriwk = 200;
/*     ZERO RPRM AND IPRM. */
  for (i__ = 1; i__ <= 20; ++i__) {
    rprm[i__ - 1] = (float)0.;
/* L10: */
    iprm[i__ - 1] = 0;
  }
    /*最適化手法の番号*/
    method =1;
    /*設計変数の数*/
    ndv = 2;
    /*制約条件の数*/
    ncon = 1;
    /*上限制約，下限制約の入力*/
    i__1 =ndv;
      for(i__=1;i__<=i__1;++i__){
            xl[i__-1] = (float)0.0;
            xu[i__-1] = (float)1.0;
      }
    /*初期点の入力（初期点は必ず制約条件内に入るようにする)*/
    x[0] = (float)0.4;
    x[1] = (float)0.2;

    /*どの程度の出力をするか*/
    iprint =1;

    /*最大化か最小化のどちらか（最大化=1，最小化=-1)*/
    minmax = -1;
    info = 0;
L100:
  dot_(&info, &method, &iprint, &ndv, &ncon, x, xl,
       xu, &obj, &minmax, g, rprm, iprm, wk, &nrwk,
       iwk, &nriwk);
  if (info == 0) {
     s_stop("", 0L);
real *obj,*x,*g;
{
  *obj=-2.0*x[0]+x[1];
  g[0]=(float)(pow((double)x[0],(double)2.0)) -x[1];
  return 0;
}
```

3.3 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．扱うことのできる手法を Table 6 に示す．

Table 6　DOT による使用アルゴリズム

手法（制約条件あり）	修正可能方向法, 逐次線形計画法, 逐次 2 次計画法
手法（制約条件なし）	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

- 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，この場合，DOT,DDOT では各変数ごとの制約条件は含まれない．つまり $0 \leq x_1 \leq 1$ といった制約条件は制約条件数に含まれない．勿論，$2x_1 + 3x_2 \geq 0$ 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 Table 7 に示すように，1〜7 の 7 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．

- 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

Table 7　iprint parameter

iprint	出力
0	no output.
1	internal parameters, initial information and results.
2	same plus objective function and X-vector at each iteration.
3	same plus G-vector and critical constraint numbers.
4	same plus gradients.
5	same plus search direction.
6	same plus set IPRM(11)=1 and IPRM(12)=1.
7	same except set IPRM(12)=2.

  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．

$$x_2 \geq -0.5x_1 + 3 \tag{13}$$

  (1) ならば, 以下の (2) のように変換し,

$$0 \geq -x_2 - 0.5x_1 + 3 \tag{14}$$

  (2) の形にしたところで,

$$g[0] = -x[1] - 0.5x[0] + 3 \tag{15}$$

  (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．

3.4 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．

3.5 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

参考文献

1) DOT USERS MANUAL Version 5.0, http://www.vrand.com/