最適化ゼミ追加マニュアル For DOT

作成者 長谷　佳明

第 I 部

DOT

1　DOT を取り上げるわけ

私たちの研究室では最適化問題に対して様々なアプローチがなされているが,DOT は最適化問題の研究者ならば知っておくべき世界的に有名なソフトウェアである. その優れた性能から, 自己の開発した最適化アルゴリズムに対する**一つの尺度**として用いることができる. 尺度とは, 例えば車の世界ならば, 他社の A 車よりも我社の B 車は○○の性能で優れているなどと表現することと似ている. つまり最適化問題であるなら, 同じ問題を DOT と自己の作成したプログラムを比較することで, 自己のアルゴリズムがいかに優れているか如実に説明することができる.

DOT は高速で優れた最適化ソフトェアである.

2　DOT とは？

2.1　DOT の特徴

DOT とは,*Design Optimization Tools* の略である. 最適化ソフトェア DOT 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. 図で各モジュールの関係をあらわすと,Fig.1.

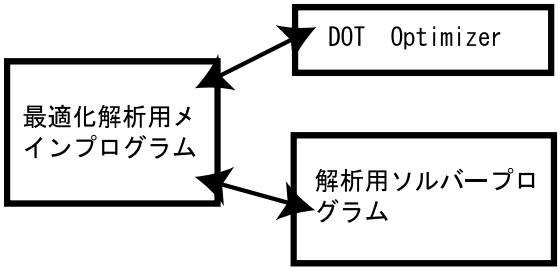

Fig. 1　DOT の解析の様子

2.2　DOT で使用しているアルゴリズム (参考程度に挙げる)

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

以上のように,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)

3 DOT の使用法

ここでは, 実際には DOT をどのように使用するのか説明する.

3.1 DOT のソース構成とコンパイル

DOT のソースは, 二種類存在し, それは, 計算精度として float 型を使用しているか,double 型を使用しているかの違いである. 前者が DOT であり *main.c*，*dot1.c*，*dot2.c*，*dot3.c*，*dot4.c*，*dot5.c*，*dot6.c* から成立し, 後者が DDOT であり,*main.c*，*ddot1.c*，*ddot2.c*，*ddot3.c*，*ddot4.c*，*ddot5.c*，*ddot6.c* から成立している. メインルーチンは main.c の中に存在している MAIN __ () に書かれており, ユーザの変更する部分も基本的にはここだけである. 変更とは, 目的関数, 制約条件などの問題に固有のパラメータの入力である. この入力は無駄であるので, これをファイルからの入力に改良したバージョンが, 改良版がある. それは,DOT の場合ならば,*main2.c*，*o__file.c*，*dot1.c*，*dot2.c*，*dot3.c*，*dot4.c*，*dot5.c*，*dot6.c* + *input.txt* であり,DDOT ならば,*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, それを基本的にはコンパイルするだけで OK である. しかし, コンパイルの際には,DOT の中で使用されている関係上,「f2c.h」, つまり,「f2c」が使用できる環境であることが前提条件となる. この「f2c」パッケージは, コンパイル時に Fortran(g77) のプログラムを c(gcc) プログラムとして変換し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.

3.2 出来上がるプログラム

コンパイルが成功すると, 一つの実行ファイルが作成される, それは,DOT に対しては,*dot* であり,DDOT に対しては,*ddot* である.

3.3 DOT プログラムの変更点

実際使用するにあたり必要プログラム変更点は, 変数値 (初期設定, 各設計変数の制約条件など), 目的関数 (制約条件) の 2 点が挙げられる. つまり, これらは, 解くべき問題に固有のパラメータであるので毎回変更が必要である. ただし, これらは改良プログラムの方では, 先にも述べたとおりファイルからパラメータを呼び込ませる形に変更されている.

3.4 DOT プログラムの変更点とパラメータの対応

method 最適化手法を指定する変数である.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」で OK である.

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 $0 \leqq x1 \leqq 1$ といった制約条件は, この場合の制約には含まない. 勿論，$2x_1 + 3x_2 > 0$ といった制約条件は含まれ, この式の本数をこの数値とする.

iprint DOT,DDOT では，3 つのレベルの出力形式に分類分けされそれぞれの出力形式を指定することができる. 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. ただし, 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 は各設計変数の値を表す. そして,xl とは設計変数の下限制約，xu と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す. 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する. これを怠ると, 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る.

g 制約条件を関数化したものである.obj 同様，関数 eval __ () により具体的な計算が行われる. 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. つまり, もし式の変換としては, 以下のように設定することになる.

$$x_2 \geq -0.5x_1 + 3 \tag{1}$$

ならば, 一度以下のように変更し,

$$0 \geq -x_2 - 0.5x_1 + 3 \tag{2}$$

この形にしたところで,

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

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, その値が負となるとき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.

改良版 (main2.c) では，基本的な変数は全て入力式に変更されいる. 具体的には，最適化手法 (method)，変数の数 (ndv)，制約条件の数 (ncon)，出力形式 (iprint)，各変数の初期点 (x)，各変数の下限制約 (xl) & 上限制約 (xu) が入力式変数として入力することが出来る. そして, その入力用ファイルとして input.txt が用意されている. ただし，関数，制約条件は入力式になっていないので, 原本の場合と同様 main.c の eval __ () を直接書き直す必要がある.

3.5 注意点

こちらのデフォルトでは, 設計変数の上限数は 100，制約条件の上限数は 50 に設定してある. 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)，xu(設計変数の上限制約) を宣言時に, 必要に応じて, より多くの配列を確保するよう変更する必要がある. ただし，あまりにも多くの設計変数，制約条件は他の配列変数にも影響を及ぼす. その場合には，rprm，wk などの変数配列も変更してやる必要がある.

4 参考プログラム

問題としては,

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

$$x_2 \geq -0.5x_1 + 3 \tag{4}$$

$$x_2 \geq -2x_1 + 6 \tag{5}$$

である.

最適解は,(R_1, R_2) として,$(2, 2)$ である.

この制約条件を図示すると以下のようになる.

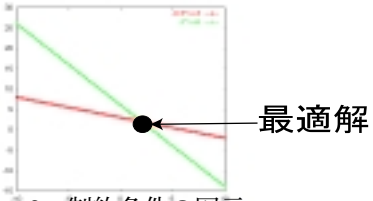

Fig. 2 制約条件の図示

```
--------------------------------------------------------------------------------
/****************
double 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 x+y に関する最適化を行っている．DDOT を使用している．
************/
/* equal.f -- translated by f2c (version 19971204).
   You must link the resulting object file with the libraries:
-lf2c -lm   (in that order)
*/
#include "f2c.h"
#include "math.h"

extern void o_file(integer *,integer *,integer *,integer *,double *,double *,double *);
/*     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 double rprm[20];
    static integer nrwk;
static double g[50];/*制約条件の MAX を 50 に設定*/
    static integer i__;
static double x[100];/*設計変数の数を MAX100 に設定*/
static integer nriwk;
static double wk[800],xl[100],xu[100];
    static integer method, minmax, iprint;
    static double 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] = (double)0.;
/* L10: */
iprm[i__ - 1] = 0;
    }
```

```
/*最適化手法の番号*/
method =1;

/*設計変数の数,R1,R2 の二つ*/
ndv = 2;

/*制約条件の数, 式としては二つ*/
ncon = 2;

/*上限制約, 下限制約の入力今回は, 一様,0 ≦ R1,R2 ≦ 10 としている*/
i__1 =ndv;
for(i__=1;i__<=i__1;++i__){
xl[i__-1] = (double)0.0;
xu[i__-1] = (double)10.0;
}

/*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)*/
/*初期点として (9.0,6.0) を設定*/
x[0] = (double)9.0;
x[1] = (double)6.0;

/*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)*/
/*最小化に決定*/
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);
    }
    eval_(&obj, x, g);
    goto L100;
} /* MAIN__ */

/* Subroutine */int eval_(obj,x,g)
double *obj,*x,*g;
{
/*目的関数*/
*obj = x[0]+x[1];
/*制約条件*/
g[0] = (float)(-x[0]-0.5*x[1]+3);
g[1] = (float)(6-2*x[1]-x[0]);
return 0;
}
```

---

5 結果の確認

結果データについて簡単に触れておく. しかしながら, これ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,iprint = 1でも表示されることが保証されているデータである.

INITIAL VARIABLES AND BOUNDS 「LOWER BOUNDS ON THE DECISION VARIABLES」: 設計変数の下限値 「DECISION VARIABLES」: 設計変数の初期値 「UPPER BOUNDS ON THE DECISION VARIABLES」: 設計変数の上限値

INITIAL FUNCTION VALUES 「OBJ」:目的関数の初期値

OPTIMIZATION RESULTS 「OBJECTIVE」: 得られた最適解 「DECISION VARIABLES」: 得られた最適解の設計変数の各値

6 問題

以下の問題を解いてください.

目的関数は,

$$\mathrm{f(x,y)=x+y} \quad \rightarrow \quad \mathrm{min}$$

制約条件は, $0 \leqq \mathrm{x}, 0 \leqq \mathrm{y}$

$$x^3 - 6x^2 + 11x - 1 \geq 0 \tag{6}$$

$$-0.5x + 6 \geq 0 \tag{7}$$

図でこれらをあらわすと,Fig.3.

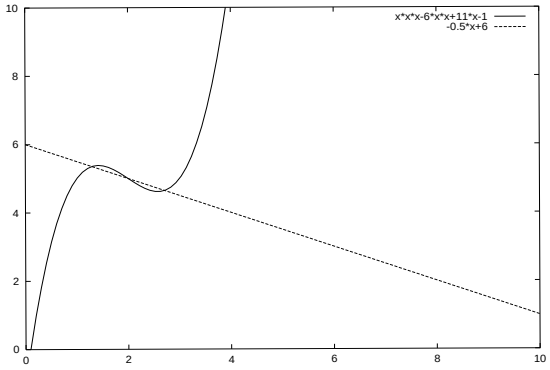

Fig. 3 問題を図示すると

7 特別に協力していただいた方

Supported By S.Watanabe From ISDL.