解説

MATLABクローンによる大域的最適化(2)
—Octaveで作る改訂単体法—

久野　誉人

5. 前回の宿題

前回に出した宿題はすんなりできただろうか？

宿題1. 線形計画問題

$$\left|\begin{array}{ll} \text{最大化} & \mathbf{c}^\top\mathbf{x} \\ \text{条　件} & \mathbf{Ax}\leq\mathbf{b},\ \mathbf{x}\geq\mathbf{O} \end{array}\right. \tag{1}$$

を解くための改訂単体法をOctaveかMATLABを使ってプログラム化しなさい．ただし，$\mathbf{A}\in\mathbb{R}^{m\times n}$，$c\in\mathbb{R}^n$とし，$\mathbf{b}\in\mathbb{R}^m$の成分はすべて非負とする．■

実は，これと同じ内容の宿題を毎年4月，筆者の研究室の学生たちにアルゴリズムを単体法に限定しないで出題している．7月の夏休み第一週，学生たちの作ってきたプログラムを1台のコンピュータに集め，Octave上でランダムに生成した様々な大きさの問題(1)を解いて計算時間を競い合うのだが，優勝者はその夜の打ち上げで他の参加者から奢ってもらえる約束だ．参加資格は学生に限っていないので，もちろん筆者も参加するのだが，残念ながら一度も優勝したことがない．これは筆者が手を抜いているわけではなく，OctaveやMATLABを使うと

(a) 教科書に書かれたプログラムの高速化に係わる常識やノウハウが必ずしも通じない，

(b) プログラミングに慣れない学生がてこずるはずの行列演算も簡単に，しかも高速に実行できる

ことが大きな原因である，と自分を慰めることにしている．そんなわけで，これから説明する宿題の解答は必ずしもベストなものといえないし，もっと賢明なやり方があることも否定しない．

すでに述べたとおり，スラック変数を基底変数$\mathbf{x}_B$，元の変数を非基底変数$\mathbf{x}_N$とし，単位行列を$\mathbf{I}\in\mathbb{R}^{m\times m}$で表すことにして

$$\mathbf{B}=\mathbf{I},\ \mathbf{N}=\mathbf{A},\ \mathbf{c}_B=\mathbf{O}^\top,\ \mathbf{c}_N=\mathbf{c}^\top$$

と置くだけで，(1)の実行可能な辞書

$$\left|\begin{array}{l} \mathbf{x}_B=\mathbf{B}^{-1}\mathbf{b}-\mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N \\ z=\mathbf{c}_B\mathbf{B}^{-1}\mathbf{b}+(\mathbf{c}_N-\mathbf{c}_B\mathbf{B}^{-1}\mathbf{N})\mathbf{x}_N \end{array}\right. \tag{2}$$

が得られる．したがってフェイズIを飛ばして，ただちに改訂単体法のフェイズIIを実行することが可能だ．釈迦に説法になるかもしれないが，念のためにフェイズIIのアルゴリズム（例えば，文献[1, 3, 6]を参照）を書いておこう：

アルゴリズム1.

ステップ0. $\mathbf{x}_B:=\mathbf{b}$, $\mathbf{x}_N:=\mathbf{O}$とおく．

ステップ1. 連立1次方程式$\mathbf{yB}=\mathbf{c}_B$を解いて$\mathbf{y}$を求め，被約費用$\bar{\mathbf{c}}_N:=\mathbf{c}_N-\mathbf{yN}$を計算する．$\bar{\mathbf{c}}_N\leq\mathbf{O}$ならば終了（$\mathbf{x}_B$，$\mathbf{x}_N$が最適解）．そうでなければ，$\bar{\mathbf{c}}_N$から値が正となる第$s$成分を選ぶ．

ステップ2. $\mathbf{N}$の第s列を$\mathbf{a}$とし，$\mathbf{Bd}=\mathbf{a}$を解いて$\mathbf{d}$を求める．$\mathbf{d}\leq\mathbf{O}$ならば終了（問題は非有界）．そうでなければ，$\bar{\mathbf{b}}:=\mathbf{x}_B-t\mathbf{d}\geq\mathbf{O}$を満たす最大の$t$を求め，$\bar{\mathbf{b}}$から値ゼロの第$r$成分を選ぶ．

ステップ3. $\mathbf{x}_B:=\bar{\mathbf{b}}$, $\mathbf{x}_N$の第s成分の値をtとおき，$\mathbf{B}$の第r列と$\mathbf{N}$の第s列$\mathbf{a}$を入れ換えるピボット演算を行う．これに合わせて$\mathbf{c}_B$，$\mathbf{c}_N$，$\mathbf{x}_B$，$\mathbf{x}_N$の成分を入れ換えたのち，ステップ1にもどる．■

この記述自体，ステップ1から3までのループで構成されており，「ループは可能な限り使うな」というOctaveによる効率的な計算の鉄則に反するが，残念ながら単体法に限らず，解を反復して改善する最適化アルゴリズムには少なくとも一つのループが必要で，これを使わないことにはプログラムにならない．

6. 二つの連立1次方程式

改訂単体法の各反復で最も手間のかかるのが，ステップ1の$\mathbf{yB}=\mathbf{c}_B$とステップ2の$\mathbf{Bd}=\mathbf{a}$の二つの連立1次方程式の求解である．これらを毎回まともに解

くの　たかひと
筑波大学 大学院システム情報工学研究科
〒305-8573 つくば市天王台1-1-1

いていたのでは非効率極まりないアルゴリズムとなるところだが，BTRAN（backward transformation），FTRAN（forward transformation）と呼ばれる巧妙な計算テクニックのお陰で，改訂単体法は誕生から50年以上を経た今も現役として活用されている．

6.1 基底行列のイータ分解

ステップ3で$\mathbf{B}$のr列と$\mathbf{N}$のs列$\mathbf{a}$を入れ換えてできる新しい基底行列$\tilde{\mathbf{B}}$は，$\mathbf{Bd}=\mathbf{a}$が成り立つことから$\mathbf{e}_r$を第r基本列ベクトルとして

$$\tilde{\mathbf{B}}=\mathbf{B}+(\mathbf{a}-\mathbf{B}\mathbf{e}_r)\mathbf{e}_r^{\top}=\mathbf{B}[\mathbf{I}+(\mathbf{d}-\mathbf{e}_r)\mathbf{e}_r^{\top}]$$

のように書くことができる．ここで$\mathbf{E}=\mathbf{I}+(\mathbf{d}-\mathbf{e}_r)\mathbf{e}_r^{\top}$と表すことにすると，$\mathbf{E}$は単位行列の第$r$列だけを$\mathbf{d}$に置き換えた**イータ行列**（Eta matrix）で，各行の非ゼロ成分は高々二つにすぎない．この方法で基底行列の更新を続ければ，初期基底行列は単位行列$\mathbf{I}$なのでk回めの反復後には

$$\mathbf{B}_k=\mathbf{E}_1\mathbf{E}_2\cdots\mathbf{E}_k$$

のように基底行列$\mathbf{B}_k$がk個のイータ行列$\mathbf{E}_1, ..., \mathbf{E}_k$の積に**イータ分解**（Eta factorization）される．したがって，次の反復のステップ1で$\mathbf{y}\mathbf{B}_k=\mathbf{c}_B$を解くときには

$$\mathbf{y}_k\mathbf{E}_k=\mathbf{c}_B,\ \mathbf{y}_{k-1}\mathbf{E}_{k-1}=\mathbf{y}_k,\ ...,\ \mathbf{y}_1\mathbf{E}_1=\mathbf{y}_2,$$

の順（BTRAN）に$\mathbf{y}_k, \mathbf{y}_{k-1}..., \mathbf{y}_1$を計算し，またステップ2で$\mathbf{B}_k\mathbf{d}=\mathbf{a}$を解くときには

$$\mathbf{E}_1\mathbf{d}_1=\mathbf{a},\ \mathbf{E}_2\mathbf{d}_2=\mathbf{d}_1,\ ...,\ \mathbf{E}_k\mathbf{d}_k=\mathbf{d}_{k-1},$$

の順（FTRAN）に$\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_k$を計算することで，それぞれの解$\mathbf{y}=\mathbf{y}_1$，$\mathbf{d}=\mathbf{d}_k$を求めることができる．係数行列はイータ行列であり，分解された各方程式を解くには$O(m)$の手間しかかからない．したがってkが大きくなければ，$\mathbf{y}\mathbf{B}_k=\mathbf{c}_B$，$\mathbf{B}_k\mathbf{d}=\mathbf{a}$をガウスの消去法で$O(m^3)$の手間をかけて愚直に計算するよりもずっと効率がよい．また，反復回数kが大きくなりすぎたときには$\mathbf{B}_k$を置換行列$\mathbf{P}_k$と三角行列$\mathbf{L}_k$，$\mathbf{U}_k$に**再分解**（refactorization）し，再び

$$\mathbf{B}_{k+l}=\mathbf{P}_k\mathbf{L}_k\mathbf{U}_k\mathbf{E}_{k+1}\mathbf{E}_{k+2}...\mathbf{E}_{k+l}$$

と更新を続ければよい．行列$\mathbf{L}_k$，$\mathbf{U}_k$はともにm個のイータ行列の積として表せるので，再分解からの反復回数lが大きくなければ，$\mathbf{y}\mathbf{B}_{k+l}=\mathbf{c}_B$も$\mathbf{B}_{k+l}\mathbf{d}=\mathbf{a}$もやはりBTRAN，FTRANによって速やかに解を求められることになる．

改訂単体法で効率の鍵を握るBTRAN，FTRANは，CやFortranなどの普通の言語を使って実装したときには期待どおりにすばらしい効果を発揮してくれる．ところが，OctaveやMATLABでこれらを実装しようとすると，例の鉄則の壁に真向からぶち当たってしまう．例えばFTRANならば，それまでに生成された$\mathbf{E}_1, \mathbf{E}_2, ..., \mathbf{E}_k$のイータ列とその列番号をそれぞれ格納する行列Dとベクトルを用意して，

```
d=a;
for j=[1:k]
    E=eye(m); E(:,R(j))=D(:,J);
    dt=E\d; d=dt;
endfor
```

のようなループの入ったプログラムを書かざるをえない．この3行めで組込み関数eye()の生成するm次単位行列Eは，R(j)列が行列Dのj列に置き換えられてイータ行列$\mathbf{E}_j$となり，次の行で方程式$\mathbf{E}_j\mathbf{d}_j=\mathbf{d}_{j-1}$が解かれる仕組みである．簡潔ではあるものの，このループを単体法の反復ごとに繰り返すプログラムがOctaveによって効率よく働いてくれるはずのないことは，前回の観察から容易に想像がつくだろう．

6.2 Octaveではどうする？

ならば，いっそのことd=B\aとやってもOctaveはFTRANのプログラムよりも高速に処理してくれるかもしれない．しかし，それでは毎回，基底行列$\mathbf{B}$の逆行列$\mathbf{B}^{-1}$を一から計算するに等しく，「改訂」のありがたみがない．そこで$\mathbf{B}$ではなく$\mathbf{B}^{-1}$の方を保持し，これをSherman-Morrison-Woodburyの**ランク・ワン更新**（rank one update）（例えば，文献[4]を参照）で次の基底行列$\tilde{\mathbf{B}}$の逆行列$\tilde{\mathbf{B}}^{-1}$に更新することにしよう．

イータ行列$\mathbf{E}$を介して$\mathbf{B}$と$\tilde{\mathbf{B}}$の間には$\tilde{\mathbf{B}}=\mathbf{B}\mathbf{E}$の関係があることは見たとおりだが，両辺の逆行列をとると$\tilde{\mathbf{B}}^{-1}=\mathbf{E}^{-1}\mathbf{B}^{-1}$になり，$\mathbf{E}$の逆行列さえ求まれば$\mathbf{B}^{-1}$を$\tilde{\mathbf{B}}^{-1}$に更新できることが分かる．計算すれば容易に確かめられるが，$\mathbf{E}^{-1}$も単位行列とはr列だけが異なるイータ行列で，d_rを$\mathbf{d}$の第r成分として

$$\mathbf{E}^{-1}=\mathbf{I}-(1/d_r)(\mathbf{d}-\mathbf{e}_r)\mathbf{e}_r^{\top}$$

と書くことができる．したがって

$$\tilde{\mathbf{B}}^{-1}=\mathbf{B}^{-1}-(1/d_r)(\mathbf{d}-\mathbf{e}_r)\mathbf{e}_r^{\top}\mathbf{B}^{-1}$$

となり，Octaveで処理しやすいように$\mathbf{d}$の第r成分d_rを-1に置き換えた$\hat{\mathbf{d}}$を使って，さらに変形すると

$$\tilde{\mathbf{B}}^{-1}=(\mathbf{I}-\mathbf{e}_r\mathbf{e}_r^{\top})\mathbf{B}^{-1}-(1/d_r)\hat{\mathbf{d}}\mathbf{e}_r^{\top}\mathbf{B}^{-1}$$

になる．つまり，$\mathbf{B}^{-1}$の第r行をすべてゼロにした行列から，列ベクトル$(1/d_r)\hat{\mathbf{d}}$と$\mathbf{B}^{-1}$の第r行との積を差し引けば$\tilde{\mathbf{B}}^{-1}$が得られるのである．この更新には$O(m^2)$の基本演算しかかからないので，$\mathbf{y}=\mathbf{B}^{-1}\mathbf{c}_B$と

$\mathbf{d}=\mathbf{B}^{-1}\mathbf{a}$ の計算を合わせても二つの方程式は $O(m^2)$ で処理できることになる．具体的には，$\mathbf{B}^{-1}$ を Bi で表して

```
dr=d(r); d(r)=-1.0; d/=dr;
br=Bi(r, :); Bi(r, :)=0.0; Bi-=d*dr;
```

とすれば Bi は $\tilde{\mathbf{B}}^{-1}$ に更新される．

ところで，イータ分解に再分解が必要となる第一の理由は $\mathbf{yB}=\mathbf{c}_B$，$\mathbf{Bd}=\mathbf{a}$ の求解で反復に比例する手間がかかるためだが，他に $O(m)$ で増加するイータ行列 $\mathbf{E}_1, ..., \mathbf{E}_k$ の記憶量，累積する丸め誤差の二つを解消するためでもある．80年代の名著[1]によると，大規模問題に対して当時，20反復に1回程度の高頻度で再分解が行われていたようだ．しかし，現在はパソコンでも当時のワークステーションよりずっと大きな主記憶を持っているし，そもそもランク・ワン更新で保持しなければならないのは常に同じサイズ $m\times m$ の行列 $\mathbf{B}^{-1}$ だけである．しかも，$\mathbf{yB}=\mathbf{c}_B$，$\mathbf{Bd}=\mathbf{a}$ に対する求解の手間は反復の多寡にかかわらず $O(m^2)$ で，これも変わらない．したがって，気にしなければならないのは丸め誤差のみで，再分解の頻度は20年前よりもはるかに少なくて大丈夫だろう．また再分解といっても，ここで欲しいのは $\mathbf{L}$ と $\mathbf{U}$ ではなくて $\mathbf{B}^{-1}$ そのものであるが，Octave の組込み関数 inv() を使えば Bi=inv(B) のように造作もなく求められる．

7. Octave 版改訂単体法

改訂単体法のプログラム化で最も厄介な課題がかたづいたので，残りの部分のプログラム化をアルゴリズム1のステップを追いながら，Octave の便利な道具を使って解説していくことにしよう．

ステップ0. 入力を $m\times n$ 行列 A，m 次列ベクトル b，n 次行ベクトル c の三つとして，

```
N=[1 : n]; B=[n+1: n+m];
x=[zeros(n, 1); b]; cc=[c, zeros(1, m)];
Bi=eye(m); AA=[A, Bi];
```

のように下ごしらえする．行列 AA は辞書(2)の $\mathbf{N}$ と $\mathbf{B}$ を並べたもので，本当は AA=[A, eye(m)] とすべきだが，この段階ではまだ基底行列もその逆行列 Bi も単位行列であるので eye(m) の呼び出しを省いている．これ以降，$\mathbf{B}$ は AA(:, B)，$\mathbf{N}$ は AA(:, N) で，$\mathbf{c}_B$ と $\mathbf{c}_N, \mathbf{x}_B, \mathbf{x}_N$ もそれぞれ cc(B)，cc(N)，x(B)，x(N) で参照することができる．また，丸め誤差を考慮してゼロの代わりに用いる 10^{-8} 程度の小さな値 Z8=1.0e-8 もこのステップで準備しておこう．

ステップ1. ランク・ワン更新した基底行列の逆行列 Bi が使えるはずなので $\mathbf{yB}=\mathbf{c}_B$ の解は y=cc(B)*Bi で求め，被約費用 $\bar{\mathbf{c}}_N$ を

```
rc=cc(N)-y*AA(:, N);
```

で計算する．同じことを for ループで行えば

```
for j=[1: n]
    rc(j)=cc(N(j))-y*AA(:, N(j));
endfor
```

のように書けるが，もちろんこれを使うべきではない．ただ，ここで注意したいのは rc(j) が x(j) でなく，x(N(j)) の被約費用である点だ．ピボット列の選択は，ポピュラーな最大係数規則を用いることにしよう：

```
[z, s]=max(rc);
```

ここで z にベクトル rc の最大成分の値が代入されるが，それが Z8 以下ならば現在の x(1: n) が最適解である．そうでなければ，値が z の s 番めの成分に対応する非基底変数 x(N(s)) が基底に入る候補となる．

ステップ2. 繁雑なので ns=N(s) とおくことにして，$\mathbf{Bd}=\mathbf{a}$ の解を d=Bi*AA(:, ns) で求める．有界性の判定には，まず

```
D=find(d>Z8);
```

によって値が Z8 よりも大きな d の成分の行番号を D に格納する．そして組込み関数 isempty() を使って isempty (D) が1か0，つまり "Is D empty?" をチェックする．もしも "yes" ならば，d の成分がすべて Z8 以下であり，問題は非有界である．そうでなければ，

```
[t, r]=min(x(B(D))./d(D)); r=D(r);
```

によって基底から掃き出す変数 x(B(r)) を特定する．

ステップ3. ピボット(r, s)が定まったので，あとはこれを中心にピボット演算を行うだけだ．その前に，ステップ2で求めた d と t を使って解を更新しよう：

```
x(B)-=t*d; x(ns)=t;
```

ピボット演算は

```
N(s)=B(r); B(r)=ns;
```

で完了する．このあと，節6.2に説明した方法で基底行列の逆行列 Bi を更新すればよい．

7.1 関数 simplex()

以上のレシピをもとに改訂単体法を Octave の関数にまとめよう：

```
function[val, sol, k]=simplex(A, b, c)
                                        %% Step 0 %%
  Z8=1.0e-8; RF=512; k=0;
  [m, n]=size(A); N=[1:n]; B=[n+1:n+m];
  x=[zeros(n, 1); b]; cc=[c, zeros(1, m)];
  Bi=eye(m); AA=[A, Bi];

  opt=0;
  while opt==0
                                        %% Step 1 %%
    y=cc(B)*Bi; rc=cc(N)-y*AA(:, N);
    [z, s]=max(rc);
    if z<=Z8
      opt=1; val=y*b; sol=x(1:n);
      else
                                        %% Step 2 %%
      ns=N(s); d=Bi*AA(:, ns);
      D=find(d>Z8);
      if isempty(D)
        opt=-1; val=Inf; sol=x(1:n);
      else
                                        %% Step 3 %%
        [t, r]=min(x(B(D))./d(D)); r=D(r);
        x(B)-=t*d; x(ns)=t;
        N(s)=B(r); B(r)=ns; k++;
        if rem(k, RF)
                                %% Rank one update %%
          dr=d(r); d(r)=-1.0; d/=dr;
          br=Bi(r, :); $Bi(r, :)=0.0;
          Bi-=d*br;
        else
                                %% Refactorization %%
          Bi=inv(AA(:, B));
          x(B)=Bi*b; x(N)=0.0;
        endif
      endif
    endif
  endwhile
endfunction
```

この関数 simplex() に問題(1)の係数行列 A と右辺ベクトル b，費用ベクトル c を引数として与えると，(1)が有界のときには最適値 val，最適解 sol，それにプログラムの終了までに要したピボット回数 k が返される．問題(1)が非有界ならば val＝Inf となるが，Octave で Inf は $+\infty$ の意味である．記号%から右は，LaTeX と同じように無視されるので注釈と思ってよい．定数 RF は再分解の頻度を示し，特に根拠はないのだが512反復に1度の再分解を行うことにしてある．ステップ3のピボット演算ののちに k を一つ増やし，剰余を返す組込み関数 rem() を使って k が RF で割り切れたときにだけ再分解が行われる．

どうです，ループはステップ1から3を繰り返す while ループ一つで済んだでしょう．それでは，関数 simplex() がコンピュータ上で正しく動いてくれるかどうか確かめてみることにしよう．

7.2 動作確認と性能評価

とりあえず，線形計画法の適当な教科書から答の分かっている例題を選んで simplex() に解かせてみよう．例えば，文献[1]の2章にある例題：

$$\left|\begin{array}{lrcrcl} \text{最大化} & 5x_1 & +5x_2 & + & 3x_3 & \\ \text{条　件} & x_1 & +3x_2 & + & x_3 & \leq 3 \\ & -x_1 & & + & 3x_3 & \leq 2 \\ & 2x_1 & -x_2 & + & 2x_3 & \leq 4 \\ & 2x_1 & +3x_2 & - & x_3 & \leq 2 \\ & & & & x_1, x_2\ x_3 & \geq 0. \end{array}\right. \tag{3}$$

最適解は $x_1=32/29$，$x_2=8/29$，$x_3=30/29$ で最適値は10である．ここで間違っても巡回を起こす問題例は選ばないこと．読者はすでに気づいていると思うが，simplex() には何の巡回対策も施していないので終了しない可能性がある．Octave を起動して，

```
octave:1> A=[1 3 1; -1 0 3; 2 -1 2; 2 3 -1];
octave:2> b=[3; 2; 4; 2]; c=[5 5 3];
```

とデータを打ち込み，これを simplex() に渡して

```
octave:3> [val, sol, k]=simplex(A, b, c)
val=10
sol=
      1.10345
      0.27586
      1.03448
k=3
```

のように瞬時に答が返ってくればひとまず安心だ．通常は，エラーメッセージがいくつも現れて，それを一つずつデバグすることになるが，メッセージはかなり詳細で，例えばサイズの合わない行列の積をとっていたりすると

```
error: operator *: nonconformant arguments (op 1 is 1×3, op 2 is 4×4)
error: evaluating binary operator '*' near line 10, column 13
.....
```

のように報告されるのですぐに修正できる．

バグ取りが終わって(3)が無事に解けるようになったら，今度はもう少し大きな問題でsimplex()の動作確認をしたい．何せ(3)は手で解くことを前提にしたオモチャ問題で，実際わずかk=3回の反復しかかかっておらずテストにならない．そうなると，適当なテスト問題を見つけてくる必要があるが，自分で作った問題を使っても(1)の双対問題：

$$\left|\begin{array}{ll}\text{最小化} & \mathbf{b}^\top\mathbf{y}\\ \text{条　件} & \mathbf{A}^\top\mathbf{y}\geq\mathbf{c},\ \mathbf{y}\geq\mathbf{0},\end{array}\right.$$

の基底解が$-$rcで与えられることを思い出せば，線形計画問題の最適性条件：(i)主実行可能性，(ii)双対実行可能性，(iii)双対性（あるいは相補スラック性）からsimplex()の出力の正否をチェックすることができる．その場合，simplex()の出力に被約費用rcも加えて

```
function[val, sol, k, rc]=simplex(A, b, c)
```

としておくとよいだろう．

さて，動作確認で障害が発見されなければ次は関数simplex()の性能を評価する段階だ．本格的な数値実験は読者に任せることにして，ここでは問題をランダムに生成し，いくつか解いてみるだけに留めよう：

```
octave:4> A=rand(200, 100)-rand(200, 100);
octave:5> b=ones(200, 1); c=rand(1, 100);
octave:6> [t 1, u 1, s 1]=cputime(); \
> [val, sol, k]=simplex(A, b, c); \
> [t 2, u 2, s 2]=cputime();
octave:7> t 2-t 1, u 2-u 1, k
ans=0.88600
ans=0.77600
k=309
```

計算時間を計測するため，time()に代ってcputime()を用いているが，二つめの出力（u 1, u 2)がOctave自身の作業時間で，三つめ（s 1, s 2)はOSが代行した作業の時間，両者の和が最初の出力（t 1, t 2）である．したがってu 2$-$u 1$=$0.77600秒が，この100変数200制約式の問題を解くのにOctaveがsimplex()で費した正味の計算時間になる．もう2題ほど解いてみよう：

```
octave:8> A=rand(300, 200)-rand(300, 200);
octave:9> b=ones(300, 1); c=rand(1, 200);
octave:10> [t 1, u 1, s 1]=cputime(); \
> [val, sol, k]=simplex(A, b, c); \
> [t 2, u 2, s 2]=cputime();
octave:11> t 2-t 1, u 2-u 1, k
ans=9.9850
ans=8,6190
k=1356
octave:12> A=rand(500, 300)-rand(500, 300);
octave:13> b=ones(500, 1);c=rand(1, 300);
octave:14> [t 1, u 1, s 1]=cputime();\
> [val, sol, k]=simplex(A, b, c);\
> [t 2, u 2, s 2]=cputime();
octave:15> t 2-t 1, u 2-u 1, k
ans=55.652
ans=46,207
k=2613
```

ランダムな問題を各サイズたった1題解いただけでは何とも評価しがたいが，とにかく関数simplex()を使って300変数500制約式の問題が50秒前後で解けた．商用コードに比べれば確かに見劣るかもしれないが，FortranやCを使ってこれだけの性能を引き出すにはとても40行足らずのプログラムでは無理だろう．

だいぶオタクな内容になってしまったが，今回の冒頭，5節に述べた(a)，(b)が宿題1を通して実感できたのではないだろうか．Octaveによるプログラミングは FortranやCとは根本的に異質で，同じ感覚で行うと効率のよいプログラムにならない．その一方，FotranやCによるプログラミングに慣れていなくても，線形代数に関する知識さえあれば，それがOctaveのプログラミングでは直に役立つのである．

8. 大域的最適化に向けて

さて，本稿の目標は改訂単体法のプログラム作成ではなく，それを道具に難しい非凸計画問題を解決することである．宿題1はこれくらいにして先に進もう．

難しいといっても，使えるのは関数simplex()だけなので解ける問題も自ずと限られてくる．大域的最適化問題としては初級の**分離可能凹最小化問題**（separable concave minimization problem）[2, 7]

$$\left|\begin{array}{ll}\text{最小化} & f(\mathbf{x})=\sum_{j=1}^{p}f_j(x_j)+\sum_{j=p+1}^{n}c_jx_j\\ \text{条　件} & \mathbf{Ax}\leq\mathbf{b},\ \mathbf{x}\geq\mathbf{0}\end{array}\right. \tag{4}$$

に的を絞り，その大域的最適解を求める**矩形分枝限定法**（rectangular branch-and-bound algorithm）を次回にOctaveでプログラムすることにしよう．問題(4)の制約条件はすべて線形計画問題(1)と同一であるものとし，さらに簡単のために実行可能領域

$$D=\{\mathbf{x}\in\mathbb{R}^n|\mathbf{Ax}\leq\mathbf{b},\mathbf{x}\geq\mathbf{0}\}$$

は有界で内点があることを仮定しよう．そうすると，

$$u_j=\max\{x_j|\boldsymbol{x}\in D\},\ j=1,\ldots,p,$$

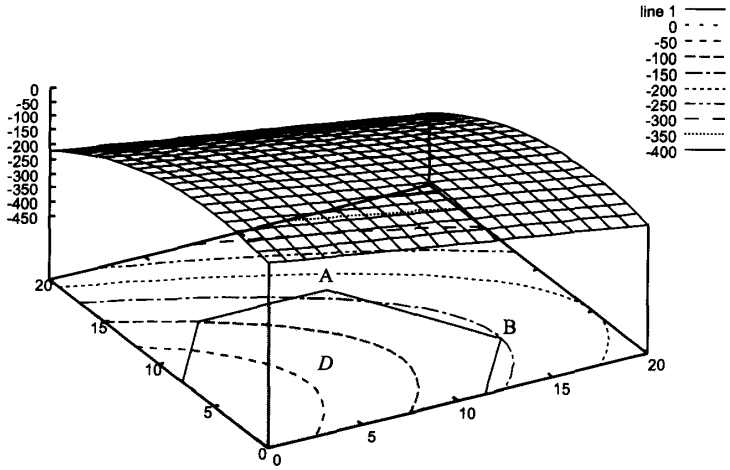

図1　問題(4)の2次元の例

はすべて正の有限値になるが，各 $j=1, ..., p$ に対して f_j は $[0, u_j]$ を含む開区間 I_j で非線形，凹関数であるものとする．つまり，どの f_j に対しても $s_j, t_j \in I_j$ ならば

$$f_j[(1-\lambda)s_j+\lambda t_j] \geq (1-\lambda)f_j(s_j)+\lambda f_j(t_j)$$

が任意の $\lambda \in [0, 1]$ で成り立つ．

問題(4)の目的関数 f は凹関数 f_j と線形関数 $\sum_{j=p+1}^{n} c_j x_j$ の和であるが，これも凹関数になる：

$$\begin{aligned} &f[(1-\lambda)\mathbf{s}+\lambda\mathbf{t}] \\ &\quad=\sum_{j=1}^{p} f_j[(1-\lambda)s_j+\lambda t_j]+ \\ &\qquad\sum_{j=p+1}^{n} c_j[(1-\lambda)s_j+\lambda t_j] \\ &\quad\geq(1-\lambda)\sum_{j=1}^{p}[f_j(s_j)+\sum_{j=p+1}^{n} c_j s_j]+ \\ &\qquad\lambda\sum_{j=1}^{p}[f_j(t_j)+\sum_{j=p+1}^{n} c_j t_j] \\ &\quad=(1-\lambda)f(\mathbf{s})+\lambda f(\mathbf{t}),\ \forall\lambda\in[0,1]. \end{aligned}$$

この不等式から

$$f[(1-\lambda)\mathbf{s}+\lambda\mathbf{t}] \geq \min\{f(\mathbf{s}), f(\mathbf{t})\},\ \forall\lambda\in[0,1],$$

が成り立つことも分かるが，これは $\mathbf{s}$ と $\mathbf{t}$ を結ぶ線分上において関数 f が $\mathbf{s}$, $\mathbf{t}$ のいずれかで最小値をとることを意味している．したがって，$\mathbf{s}$, $\mathbf{t}$ とも D から選んだ場合を頭の中に描けば，f は凸多面体 D のいずれかの端点で少なくとも局所的に最小化されるはずである．大域的最適解は局所最適解の中にあるので，このことは(4)を解くうえで非常に重要な手がかりである．白状すると，今のところ手がかりはそれがすべてといって過言でない．大域的な最適性を保証するには，すべての局所最適解の値を比較するより手だてがないのである．ところが，局所最適解の候補である D の端点の数は次元 n の指数関数であり，それを単純に列挙していては数十変数の問題であっても現実的な計算時間のうちに大域的最適解へはたどり着けない．

難しさの本質はもちろん凹関数 f_j にあり，その数 p が少なければ問題(4)は効率よく解けそうに思える．この直感は半ば正しいが，計算の複雑さからいえば $p=1$ の場合も(4)はNP困難である[5]．図1は，$n=2$, $p=1$ のときの関数 f の例

$$f(x_1, x_2)=-(x_1-5)^2-10x_2$$

を等高線とともにOctaveで（厳密にいえば，Octaveから呼び出されたGnuplotで）プロットし，後からフリーのお絵書きソフトTgif[8]を使って凸多角形 D を描き加えたものだ．したがって問題(4)のイメージを示しているが，こんなに小さな問題例にさえ点A，Bのような複数の局所最適解が存在する．高次元の問題ともなれば，その難しさは推して知るべしである．悲観的な締めくくりとなったが，次回はこの印象の払拭に努めることにしよう．

参考文献

[1] Chvátal, V., *Linear Programming*, Freeman (1983).

[2] Horst, R. and H. Tuy, *Global Optimization : Deterministic Approaches*, Springer (1993).

[3] 今野　浩,「線形計画法」, 日科技連 (1987).

[4] Martin, R. K., *Large Scale Linear and Integer Optimization*, Kluwer (1999).

[5] Pardalos, P. M. and S. A. Vavasis, "Quadratic programming with one negative eigenvalue is NP-hard," *Journal of Global Optimization* 1 (1991), 15-22.

[6] 田村明久, 村松正和,「最適化法」, 共立出版 (2002).

[7] Tuy, H., *Convex Analysis and Global Optimizaton*, Kluwer (1998).

[8] http : //bourbon.usc.edu:8001/tgif/