解説

MATLABクローンによる大域的最適化(3)
—Octaveはここまでできる—

久野　誉人

9. 矩形分枝限定法

Octave版改訂単体法simplex()で料理するのは，分離可能凹最小化問題[6, 7]

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

に決まった．前回，仮定したように制約条件は線形計画問題(1)と同じ，$\mathbf{A}\in\mathbb{R}^{m\times n}$で$\mathbf{b}\in\mathbb{R}^m$は非負，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ものとする．したがって，

$$u_j=\max\{x_j|\mathbf{x}\in D\},\quad j=1,\ldots,p, \tag{5}$$

はすべて正の有限値であり，各$j=1,\ldots,p$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．料理のまえにその段取りとなる**矩形分枝限定法**(rectangular branch-and-bound algorithm)について説明しておこう．拙稿[4]と重なる部分もあるが，おそらくは忘却の彼方であろうから省略せずに解説しよう．

9.1 整数計画法と同じ？

実行可能領域D上では，目的関数fに対して非線形に働く変数$\mathbf{x}_P=(x_1,\ldots,x_p)^T$が(5)で定まる矩形

$$C=[0, u_1]\times\cdots\times[0, u_p]$$

の外にはみ出ることはない．したがって，問題(4)に条件$\mathbf{x}_P\in C$を追加して

$$\mathrm{P}(C)\left|\begin{array}{ll}\text{最小化} & f(\mathbf{x})\\ \text{条　件} & \mathbf{Ax}\leq\mathbf{b},\ \mathbf{x}\geq\mathbf{0},\ \mathbf{x}_P\in C\end{array}\right.$$

を解いても(4)と同じ解が得られるはずだ．矩形分枝限定法は，int・で集合の内部を表すことにすると，

$$C=\cup_{k\in\mathcal{L}}C^k,\quad \mathrm{int}\,C^k\cap\mathrm{int}\,C^l=\emptyset,\quad k\neq l,$$

が満たされるようにCを複数の矩形$C^k(k\in\mathcal{L})$に分割し，(4)の子問題$\mathrm{P}(C^k)$を次々に解いていく方法である．矩形Cを小さなC^kに差し替えたところで，子問題$\mathrm{P}(C^k)$は元の問題(4)と本質的に同一の凹最小化問題であり，直接には解けない．そこで通常，以下のように再帰的な方法が用いられている：

アルゴリズム2.

ステップ0.　$k:=1, C^1:=C, \mathcal{L}:=\{1\}, \mathbf{x}^*:=\mathbf{0}, z^*:=f(\mathbf{x}^*)$とおく．

ステップ1[子問題選択].　$\mathcal{L}=\emptyset$ならば終了($\mathbf{x}^*$が最適解)．そうでなければ，$\mathcal{L}$から適当な添字i_kを取り出して$C:=C^{i_k}$とおく．

ステップ2[限定操作].　子問題$\mathrm{P}(C)$の下界値$\bar{z}$を計算する．$f(\mathbf{x}^k)<z^*$を満たす実行可能解$\mathbf{x}^k$が得られれば，$\mathbf{x}^*:=\mathbf{x}^k$, $z^*:=f(\mathbf{x}^k)$と更新する．$\bar{z}\geq z^*$ならば，Cを破棄してステップ1に戻る．

ステップ3[分枝操作].　Cを2つの矩形C^{2k}, C^{2k+1}に分割し，$\{2k, 2k+1\}$を$\mathcal{L}$に加える．$k:=k+1$としてステップ1に戻る．■

整数計画法の知識があれば，すぐにこのアルゴリズムが混合整数計画問題

$$\left|\begin{array}{ll}\text{最小化} & \mathbf{c}^T\mathbf{x}\\ \text{条　件} & \mathbf{Ax}\leq\mathbf{b},\ \mathbf{x}\geq\mathbf{0},\ \mathbf{x}_P\in\{0,1\}^p\end{array}\right. \tag{6}$$

を解くための分枝限定法と同様のものであることに気づくだろう．分枝操作で$\{0,1\}^p$を分割すれば，その要素はアルゴリズムの反復とともに着実に減少し，有限回の反復後に1点となって(6)の子問題は簡単な線形計画問題に帰着する．ところが，凹最小化問題(4)ではCをいくら分割しても1点にはならず，$\mathrm{P}(C)$が簡単な問題になることはない．その一方，(6)では$\{0,1\}^p$が1点になるまで分割しないと実行可能解すら見つけることは難しいが，以下で見るように(4)では限定操作の下界値$\bar{z}$を計算する過程で，副産物として反復のたびに実行可能解を得ることができる．

9.2 一般的な限定操作

矩形Cはk回めの反復で

$$C=[s_1, t_1]\times\cdots\times[s_p, t_p]$$

となっているものとしよう．目的関数の各f_jは区間

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

$[s_j, t_j]$で凹関数であり，前回にも示した通り，

$$f_j[(1-\lambda)s_j+\lambda t_j] \geq (1-\lambda)f_j(s_j)+\lambda f_j(t_j) \quad (7)$$

が任意の$\lambda \in [0,1]$で満たされる．このパラメータλを使えば$[s_j, t_j]$上の点は$x_j=(1-\lambda)s_j+\lambda t_j$のように書けるので，これを使って(7)から$\lambda$を消せば

$$f_j(x_j) \geq \frac{f_j(t_j)-f_j(s_j)}{t_j-s_j}(x_j-s_j)+f_j(s_j)$$

となる．この不等式の右辺を

$$g_j(x_j)=\frac{f_j(t_j)-f_j(s_j)}{t_j-s_j}(x_j-s_j)+f_j(s_j)$$

と定義すると，f_jとアフィン関数g_jとの間に

$$g_j(x_j) \leq f_j(x_j), \forall x_j \in [s_j, t_j] \quad (8)$$

の関係が成り立ち，$x_j \in \{s_j, t_j\}$のときには$g_j(x_j)=f_j(x_j)$となる．さらに，f_jは非線形なので

$$g_j(x_j) > f_j(x_j), \forall x_j \in I_j \backslash [s_j, t_j] \quad (9)$$

を示すこともできる．不等式(8)から直ちにわかるのは，子問題$\mathrm{P}(C)$で各f_jをg_jに置き換えた問題

$$\tilde{\mathrm{P}}(C)\left|\begin{array}{ll}\text{最少化} & g(\mathbf{x})=\sum_{j=1}^{p} g_j(x_j)+\sum_{j=p+1}^{n} C_j x_j \\ \text{条　件} & \mathbf{Ax} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0},\ \mathbf{x}_P \in C\end{array}\right.$$

を解くと，その最適値として$\mathrm{P}(C)$の下界値が手に入ることだ．問題$\tilde{\mathrm{P}}(C)$は，g_jがアフィン関数なので線形計画問題である．早速，関数simplex()を使ってみたくなるが，厄介なのは追加した条件$\mathbf{x}_P \in C$の存在である．この条件は単なる有界条件

$$s_j \leq x_j \leq t_j,\quad j=1, \ldots, p,$$

なので，**有界単体法** (bounding simplex algorithm) [1]を使えば非負条件を扱うのと変わらない手間で処理できるが，残念ながら関数simplex()にその機能は持たせていない．そのため，simplex()で処理するときには$2p$個の不等式条件を$\mathbf{Ax} \leq \mathbf{b}$の中に組み込んで処理せざるをえない．これはいかにも馬鹿げている．

9.3 関数simplex()で行う限定操作

下界値の計算で問題$\tilde{\mathrm{P}}(C)$を解く方法を最初に提案したのはFalkとSoland[2]で1969年の大昔だが，その5年後にSolandは$\tilde{\mathrm{P}}(C)$に有界条件$\mathbf{x}_P \in C$が必ずしも必要のないことを示している[5]．つまり，$\tilde{\mathrm{P}}(C)$を簡略して線形計画問題(1)とまったく同じ構造の

$$\bar{\mathrm{P}}(C)\left|\begin{array}{ll}\text{最小化} & g(\mathbf{x}) \\ \text{条　件} & \mathbf{Ax} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\end{array}\right.$$

を解けばよいのである．この問題$\bar{\mathrm{P}}(C)$の最適解$\mathbf{x}^k$をsimplex()で求めれば，子問題$\mathrm{P}(C)$の下界値は$z=g(\mathbf{x}^k)$で与えられる．問題$\tilde{\mathrm{P}}(C)$の制約条件を緩和したわけだから至極あたりまえだが，代償として$\tilde{\mathrm{P}}(C)$よりも弱い下界値しか得られない．しかし，矩形Cがどのように分割されようと$\bar{\mathrm{P}}(C)$の実行可能領域は常に不変であり，目的関数をいくらか修正すれば前の反復で解いた$\bar{\mathrm{P}}(C)$の最適解$\mathbf{x}^{k-1}$を初期基底解としてsimplex()をすぐさま実行することができる．反復のたびに線形計画問題(1)の利潤ベクトル$\mathbf{c}$に関する感度分析を行うのと同じである．また，simplex()が生成する$\bar{\mathrm{P}}(C)$の最適解$\mathbf{x}^k$は目標の問題(4)の実行可能解でもあるので，暫定値z^*に対して$f(\mathbf{x}^k)<z^*$を満たせば，暫定解$\mathbf{x}^*$の更新にも用いることができる．

利潤ベクトル$\mathbf{c}$以外のデータが共通の線形計画問題を繰り返し解くことになるので，前回作ったsimplex()のように，すべての定数，変数が関数内でローカルに定義されていてはいささか都合が悪い．そこで，行列$\mathbf{A}$や右辺ベクトル$\mathbf{b}$，基底$\mathbf{B}$，非基底$\mathbf{N}$など，simplex()が呼び出されるたびに初期化されては困るものを外部変数として別のところで定義することにし，$\mathbf{c}$を引数ccとしてsimplex()に渡せば，それに対する最適値valだけを返すように書き換えておこう：

```
function val=simplex(cc)
                                        %% Step 0 %%
  global Z8; global RF; global k;
  global m; global n; global B; global N;
  global x; global AA; global Bi; global b;

  opt=0;
  while opt==0
                                        %% Step 1 %%
    y=cc(B)*Bi; rc=cc(N)-y*AA(:,N);
    ......                                  ......

  endwhile
endfunction
```

例えば，「global x」は解を収める外部変数xの使用宣言であり，global宣言さえあればsimplex()以外からもxにアクセスできる．このわずかな修正で関数simplex()は，$\bar{\mathrm{P}}(C)$の再最適化を行って子問題$\mathrm{P}(C)$の下界値zを生成するサブルーチンとしてアルゴリズム2のプログラムから呼び出すことが可能になる．

9.4 ω分割による分枝操作

節9.1で触れたように，離散最適化の場合と違って凹最小化問題(4)に対する分枝限定法では，矩形Cをいくら分割しても一般に$\mathrm{P}(C)$が自明な最適解をもつことはない．そのため，実際に使うときには分枝が無

限に続くことのないように，ステップ2の打ち切り判定 $\bar{z}\geq z^*$ に $\epsilon>0$ の誤差を許して $\bar{z}\geq z^*-\epsilon$ とすることが多い．しかし，$\epsilon=0$ であっても，ステップ2で $\bar{\mathrm{P}}(C)$ を解くことに加え，ステップ3での C の分割に，これから紹介する **ω 分割** (ω-subdivision) を用いれば有限回の反復後に必ずアルゴリズムを終了させることができる．Soland[5]の功績は，$\tilde{\mathrm{P}}(C)$ の簡略化などよりむしろ，これを示したところにある．

問題 $\bar{\mathrm{P}}(C)$ の最適基底解 $\mathbf{x}^k$ に対して，まず

$$q\in\arg\max\{f_j(x_j^k)-g_j(x_j^k)|j=1,\ldots,p\} \tag{10}$$

を選び，x_q^k を境に区間 $[s_q, t_q]$ を $[s_q, x_q^k]$ と $[x_q^k, t_q]$ の2つに分割する．そして，新たな矩形 C^{2k} と C^{2k+1} を次のように定義する：

$$\left.\begin{aligned}C^{2k}&=[s_q, x_q^k]\times\textstyle\prod_{j\neq q}[s_j, t_j]\\ C^{2k+1}&=[x_q^k, t_q]\times\textstyle\prod_{j\neq q}[s_j, t_j].\end{aligned}\right\} \tag{11}$$

区間 $[s_q, t_q]$ は分割できないかもしれないが，$x_q^k\notin(s_q, t_q)$ ならば(8)，(9)によって $g_q(x_q^k)\geq f_q(x_q^k)$ が成り立つはずである．添字 q の選択の仕方から，$g_j(x_j^k)\geq f_j(x_j^k)$ がすべての j に対して成り立ち，

$$\begin{aligned}\bar{z}&=g(\mathbf{x}^k)\\&=\textstyle\sum_{j=1}^{p}g_j(x_j^k)+\sum_{j=p+1}^{n}c_jx_j^k\\&\geq\textstyle\sum_{j=1}^{p}f_j(x_j^k)+\sum_{j=p+1}^{n}c_jx_j^k\\&=f(\mathbf{x}^k)\geq z^*\end{aligned}$$

となって C はステップ2で破棄されていなければならない．この ω 分割を各反復で実行すると矩形 C は，$\mathbf{x}^k$ が基底解なので，凸多面体 D の端点を通って x_1 軸から x_p 軸までのいずれかに垂直な平面でしか分割されない．端点の数も座標軸の数も有限であり，C は有限回しか分割できないことになる．したがって，アルゴリズム2も有限回の反復後に終了する．結局，誤差 ϵ を使わなくて済み，終了時の $\mathbf{x}^*$ に残されるのは凹最小化問題(4)の厳密な大域的最適解である．

10. Octave 版矩形分枝限定法

問題(4)の数値的な解決は，その目的関数に含まれる凹関数 f_j の値が計算できないことには始まらない．ここでは単純に $c_j\geq0$ に対し，2次関数

$$f_j(x_j)=-c_jx_j^2,\quad j=1,\ldots,p, \tag{12}$$

によって f_j の値を与えることにするが，後から変更することは容易だ．まな板にのるのは凹2次計画問題

$$\left|\begin{aligned}&\text{最小化}\quad -\textstyle\sum_{j=1}^{p}c_jx_j^2-\sum_{j=p+1}^{n}c_jx_j\\&\text{条　件}\quad \mathbf{Ax}\leq\mathbf{b},\ \mathbf{x}\geq\mathbf{0}\end{aligned}\right. \tag{13}$$

である．さあ，料理に取り掛かることにしよう．

10.1 再帰関数による実現

凹関数 $f_j(x_j)=-c_jx_j^2$ に対して区間 $[s_j, t_j]$ で下界値を与えるアフィン関数 g_j は，$s_j\neq t_j$ ならば

$$g_j(x_j)=-c_j(s_j+t_j)x_j+c_js_jt_j$$

と書けるので，$(c_j(s_j+t_j), c_js_jt_j)^{\mathrm{T}}$ を $2\times p$ 行列 G の j 列に格納する．また，矩形 C も区間 $[s_j, t_j]$ の端点 $(s_j, t_j)^{\mathrm{T}}$ を j 列にもつ $2\times p$ の行列 C で表すことにする．こうしておけば，ステップ3で $[s_q, t_q]$ が更新されても，c_j を成分とするベクトル c を使って

```
G(1, q)=c(q) * (C(1, q)+C(2, q));
G(2, q)=c(q) * (C(1, q)+C(2, q));
```

のように $\bar{\mathrm{P}}(C)$ の目的関数 g を修正できるし，

```
zb=sum(G(2, :))
   -simplex([G(1, :), c(p+1:m+n)]);
```

によって下界値 $\bar{z}$ も計算できる．問題(13)に c_j は n 個しかないが，c(n+1:m+n) はゼロ・ベクトルである．問題 $\bar{\mathrm{P}}(C)$ の最適解 $\mathbf{x}^k$ は simplex() が算出して外部変数 x に格納するので，これを使えばベクトル

```
fx=c(1:p)'.*x(1:p).*x(1:p);
```

の第 j 成分が $-f_j(x_j^k)$ である．したがって，$\mathbf{x}^k$ における(13)の目的関数値は

```
tz=-sum(fx)-c(p+1:n) *x(p+1:n);
```

に等しく，また下界値 zb が暫定値よりも小さいときには ω 分割の(10)を

```
[v, q]=max(G(1, :)'.*x(1:p)-G(2, :)'-fx);
```

で計算すればよい．このときに必要なのは q だけだが，x(q) で区間 $[s_q, t_q]$ を分割，更新し，(11)で定義される新たな矩形2つに以上の手続きを再び施す．

便利な道具をさらにもう一つ紹介しよう．うれしいことに，Octave では関数の再帰呼び出しが可能である．再帰を使えば，上の手続きの繰り返しがコンパクトな次の関数 recursion() で実現できる：

```
function recursion(r, q, w, G, C)

  global Z8; global m; global n; global p; global c;
  global x; global iz; global ix; global num;
                                        %% Step 2 %%
  ++num;
  C(r, q)=w;
  G(1, q)=c(q) * (C(1, q)+C(2, q));
  G(2, q)=c(q) * (C(1, q)+C(2, q));
  zb=sum(G(2, :))-simplex([G(1, :), c(p+1:m+n)]);
  fx=c(1:p)'.*x(1:p).*x(1:p);
```

```
    tz=-sum(fx)-c(p+1:n)*x(p+1:n);
    if(tz<iz)
        iz=tz; ix=x(1:n);
    endif
                                        %% Step 3 %%
    if(iz-zb>Z8)
        [v, q]=max(G(1, :)'.*x(1:p)-G(2, :)'-fx);
        w=x(q);
                                        %% Step 1 %%
        recursion(2, q, w, G, C); recursion(1, q, w, G, C);
    endif
endfunction
```

アルゴリズム2のステップ1-3がどの部分に対応するかを示すのは，再帰的に記述したこの関数recursion()では難しいが，敢えて記せば%で始まる注釈のようになる．ステップ1の子問題選択は，再帰呼び出しで最も簡単に実装できる**深さ優先則** (depth-first rule) を採用した．したがって，子問題の中から常に最も新しく生成されたものが選択される．変数ixは暫定解$\mathbf{x}^*$，izはその目的関数値$z^*=f(\mathbf{x}^*)$である．下界値tzとizの比較による打ち切り判定には外部で定義する誤差Z8=1.0e-8を許しているが，これは丸め誤差のイタズラを防ぐためであり，理論的には必要ない．また，recursion()の中で$\bar{\mathrm{P}}(C)$の目的関数gと矩形Cの情報をそれぞれ収めた行列G, Cが再帰的に呼び出されるrecursion()に引数として渡されるが，C言語と同じでG, Cそのものではなく，その値のコピーが渡されるに過ぎない．そのため，呼び出されたrecursion()がコピーを書き換えても，呼出しから帰ったときに元のG, Cにまで変更がおよんでいる心配はない．

10.2 関数rectBB()

関数recursion()は，$\bar{\mathrm{P}}(C)$の目的関数や矩形Cに関する情報が与えられたときにステップ1-3を繰り返すだけで単独では何もできない．そこで，外部変数に必要な値を与えてrecursion()を起動させるプログラムが必要になる．それが，次に示す関数rectBB()だ：

```
function[val, sol, nsb, piv]=rectBB(A, rhs, nc, lc)
                                        %% Step 0.1 %%
    global Z8; global RF; global k; global m;
    global n; global B; global N; global x;
    global AA; global Bi; global b; global c;
    global p; global iz; global ix; global num;
    Z8=1.0e-8; RF=512; k=0;
    [m, n]=size(A); N=[1:n]; B=[n+1:n+m];
    x=[zeros(n, 1); rhs]; c=zeros(1, n+m);
    Bi=eye(m); AA=[A, Bi]; b=rhs;
    p=columns(nc); C=G=zeros(2, p);
                                        %% Step 0.2 %%
    for j=[1:p]
        c(j)=1.0; C(2, j)=simplex(c); c(j)=0.0;
    endfor
    G(1, :)=nc.*C(2, :); c(1:p)=nc; c(p+1:n)=lc;
                                        %% Branch-and-bound %%
    zb=-simplex([G(1, :), c(p+1:m+n)]); num=p+1;
    fx=c(1:p)'.*x(1:p).*x(1:p);
    iz=-sum(fx)-lc*x(p+1:n); ix=x(1:n);
    if(iz-zb>Z8)
        [v, q]=max(G(1, :)'.*x(1:p)-fx);
        w=x(q);
        recursion(2, q, w, G, C); recursion(1, q, w, G, C);
    endif
    val=iz; sol=ix; nsb=num; piv=k;
endfunction
```

関数rectBB()は，問題(13)の行列$\mathbf{A}$，ベクトル$\mathbf{b}$，それに目的関数の非線形部分の係数$(c_1, ..., c_p)$と線形部分の$(c_{p+1}, ..., c_n)$をそれぞれA, rhs, nc, lcとして渡すと，(13)の最適値val，最適解sol，関数simplex()の呼び出し回数nsb，およびピボット演算の総数pivを返す．注釈のStep 0.1で示した部分で関数simplex()やrecusion()と共有する外部変数の初期化を行い，Step 0.2で初期矩形$C^1=[0, u_1]\times\cdots\times[0, u_p]$を定めている．鉄則に反して，ちゃっかりforループを使っているが，できるだけタイトな矩形を求めるには(5)に相当するp題の線形計画問題を解かざるをえない．注釈でBranch-and-boundと書いたところから下がアルゴリズム2のステップ1-3になるが，この部分は関数recursion()と重複する．これは，最初の子問題$\mathrm{P}(C^1)$をrectBB()の中で処理しているためだが，$\bar{\mathrm{P}}(C^1)$の最適解$\mathbf{x}^1$と$f(\mathbf{x}^1)$をそれぞれ暫定解ix，暫定値izに代入してから再帰関数recursion()が呼び出される．

10.3 性能評価と考察

線形計画問題と違ってプログラムの動作確認に適当な非凸計画問題のテスト問題を捜し出すのはなかなか容易でないが，(12)で定義した凹関数f_jに線形項を足して$f_j(x)=-c_jx_j^2+d_jx_j$とすれば，問題集[3]にいくつか例題を見つけることができる．それらを解く場合

は，rectBB()，recursion()で目的関数値の計算に若干の修正が必要だ．紙幅が少なくなってきたので，動作確認は省略し，(13)のランダムに生成した問題例だけを関数rectBB()で解いてみることにしよう．

まずは，小手調べに$(m, n, p)=(50, 30, 10)$の問題を解いてみると

```
octave:1> A=rand(50, 30)-rand(50, 30);
octave:2> b=ones(50, 1); c=rand(1, 30);
octave:3> [t1, u1, s1]=cputime(); \
> [val, sol, nsb, piv]=rectBB(A, b, c(1:10), c(11:30)); \
> [t2, u2, s2]=cputime();
octave:4> t2-t1, nsb, piv
ans=0.46800
ans=291
ans=478
```

になり，0.47秒の間に関数simplex()が291回呼び出されて計478回のピボット演算を行ったことがわかる．同じデータで$(m, n, p)=(50, 30, 20)$としても，

```
octave:5> [t1, u1, s1]=cputime(); \
> [val, sol, nsb, piv]=rectBB(A, b, c(1:20), c(21:30)); \
> [t2, u2, s2]=cputime();
octave:6> t2-t1, nsb, piv
ans=8.3240
ans=9537
ans=3987
```

のように8秒強で解けた．注目すべきは，関数simplex()の呼び出し回数よりもピボット演算の回数の方がずっと少ない点だ．小規模であるとはいえ，9千もの線形計画問題$\bar{\mathrm{P}}(C)$をいちいち端から解いていては，とても8秒やそこらで(13)の大域的最適解にたどり着けそうにない．しかし，単体法の感度分析は各$\bar{\mathrm{P}}(C)$を再最適化するのに平均して0.5回に満たないピボット演算しか必要としなかったのである．

もっと大きな問題でも大丈夫そうなので，思い切って行列$\mathbf{A}$のサイズを$(m, n)=(150, 100)$にして$p=10, 15, 20$の場合を解いてみたところ，以下のような結果となった：

```
octave:7> A=rand(150, 100)-rand(150, 100);
octave:8> b=ones(150, 1); c=rand(1, 100);
octave:9> [t1, u1, s1]=cputime(); \
> [val, sol, nsb, piv]=rectBB(A, b, c(1:10), c(11:100)); \
> [t2, u2, s2]=cputime();
octave:10> t2-t1, nsb, piv
ans=9.0040
ans=401
ans=4549
octave:11> [t1, u1, s1]=cputime(); \
> [val, sol, nsb, piv]=rectBB(A, b, c(1:15), c(16:100)); \
> [t2, u2, s2]=cputime();
octave:12> t2-t1, nsb, piv
ans=27.457
ans=2972
ans=12507
octave:13> [t1, u1, s1]=cputime(); \
> [val, sol, nsb, piv]=rectBB(A, b, c(1:20), c(21:100)); \
> [t2, u2, s2]=cputime();
octave:14> t2-t1, nsb, piv
ans=198.45
ans=39413
ans=75851
```

非線形項f_jの数pの増加とともにCPU時間も加速的に増加するが，30年も前のアルゴリズムがベースのプログラムとしては大健闘ではないだろうか．実のところ，simplex()と違って関数rectBB()もrecurtion()も計算効率より記述の簡潔さを優先してある．便利な再帰呼び出しも，局所変数の生成・消去を繰り返すばかりで無駄が多く，本当に効率のよいプログラムを書きたいのであれば使うべきではない．アルゴリズム2の添字集合$\mathcal{L}$をスタックとして自前で管理すれば，プログラムのCPU時間は劇的に改善するはずだ．

11. おわりに

第1回に約束したのでOctaveそのものの高速化についても記しておこう．Octaveの心臓ともいえる2つの線形代数ライブラリーBLAS (Basic Linear Algebra Subprograms) とLAPACK (Linear Algebra PACKage) をチューン・アップする．まずはインターネットで数値計算ライブラリーの宝庫Netlib [8]にアクセスし，ATLAS (Automatically Tuned Linear Algebra Software) の最新安定版atlas 3.6.0をダウンロードしよう．圧縮されたソースatlas 3.6.0.tgzを解凍するとATLASという名のディレクトリが生成され，その下にコンパイルに必要なファイルが展開される．では，ディレクトリATLASに降りてコンパイル作業を始めよう．

最初に計算環境に合わせた設定ファイルを作るため，OSのコマンドラインから

```
$ make config
```

と入力する．ターミナルに出力されるいくつかの質問

に答えなければならないが，よほど特殊な環境でないかぎり，自動検出されるデフォルトに設定しておけば大丈夫．この作業の終りに，例えば“make install arch=Linux_P4SSE2”のように次に行うべき作業の指示が出力される．この場合，OSとしてはLinux，アーキテクチャとしてPentium 4が選択されたことを意味するが，指示にしたがって

```
$ make install arch=Linux_P4SSE2
```

と入力すれば，あとはチューン・アップされたオブジェクト・ファイル*.oのアーカイブができるのを待てばよい．この工程はかなりの時間がかかるが，“ATLAS install complete”が出力されれば終了である．アーカイブは，現在のディレクトリATLASの中のlibの下，この例ではディレクトリLinux_P4SSE2に生成される．

ディレクトリLinux_P4SSE2には7つのファイルがあるが，*.aファイルの5つがアーカイブである．これらをOSの共有ライブラリが置かれているディレクトリ/usr/libか/usr/local/libなどにコピーすればよいが，アーカイブの1つliblapack.aと同じ名前のファイルが/usr/libにも存在するので注意しなければならない．新しく生成された方のliblapack.aには元もと存在する/usr/lib/liblapack.aを補完するオブジェクト・ファイルが納められており，両者をマージして1つにする必要がある．それには，まず

```
$ mkdir tmp; cd tmp; ar x../liblapack.a; cd..
```

とやってLinux_P4SSE2の下に一時的なディレクトリtmpを作り，その中でアーカイブliblapack.aを展開する．次に，

```
$ mv /usr/lib/liblapack.a .; ar r liblapack.a tmp/*.o
```

のように/usr/libの下のliblapack.aをLinux_P4SSE2に移動させ，このアーカイブにtmpで展開したオブジェクト・ファイル*.oを追加する．この作業が終わったら，Linux_P4SSE2の下にある5つのアーカイブlibatlas.a, libcblas.a, libf 77 blas.a, liblapack.a, libtstatlas.aを/usr/libか/usr/local/libにコピーする．あとは，Octaveを再コンパイルするだけだ．バイナリーではなく，ソース・パッケージの方のoctave-2.1.57.src.rpmを，例えば/usr/src/redhat/SRPMの下で

```
$ rpmbuild --rebuild octave-2.1.57.src.rpm
```

とやれば，ATLASが自動的に組み込まれたバイナリー・パッケージoctave-2.1.57.i386.rpmができあがる．これをインストールすれば作業終了である．

チューン・アップ前のOctaveで1,000変数の連立1次方程式$\mathbf{Ax}=\mathbf{b}$を解いたところ，

```
octave:15> rand("seed", 2005); b=rand(1000, 1);
octave:16> A=rand(1000, 1000)-rand(1000, 1000);
octave:17> t=time(); A\b; time()-t
ans=1.2715
```

であった結果が，チューン・アップ後には

```
octave:1> rand("seed", 2005); b=rand(1000, 1);
octave:2> A=rand(1000, 1000)-rand(1000, 1000);
octave:3> t=time(); A\b; time()-t
ans=0.83288
```

のようになった．1行目の「rand(“seed”, 2005)」は，データを同一にする乱数の種の指定である．どうです，ますますOctaveを始めてみたくなったでしょ？

謝辞 この連載で行った研究の一部は日本学術振興会科研費，基盤研究(C)17560050の補助を受けています．また，連載の機会を与えてくださった編集委員の成蹊大学池上先生には，原稿に関する貴重なコメント，アドバイスの数々に心から感謝いたします．

参考文献

[1] Chvátal, V., *Linear Programming*, Freeman (1983).
[2] Falk, J. E. and R. M. Soland, “An algorithm for separable nonconvex programming problems”, *Management Science* 15 (1969), 550-569.
[3] Floudas, A., *et al.*, *Handbook of Test Problems in Local and Global Optimization*, Kluwer (1999).
[4] 久野誉人, “非凸計画問題≠解けない問題—分枝限定法による大域的最適化”, **オペレーションズ・リサーチ 44** (1999), 232-236.
[5] Soland, R. M., “Optimal facility location with concave costs”, *Operations Research* 22 (1974), 373-382.
[6] Horst, R. and H. Tuy, *Global Optimization : Deterministic Approaches*, Springer (1993).
[7] Tuy, H., *Convex Analysis and Global Optimization*, Kluwer (1998).
[8] http://www.netlib.org/