線型計画法
— Risa/Asir のライブラリ os_muldif.rr の関数 linprog() —

Toshio Oshima

(2026.6.26, 2026,7.29)

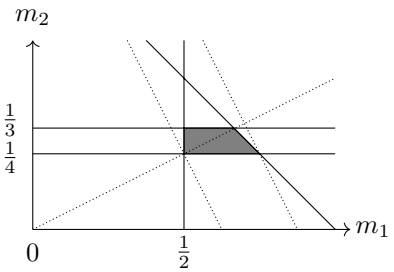
`linprog(P,E|var=`v`,verb=1,verb=1)`

- P =`[`p_1`,`p_2`,...]` : $p_1 \geq 0$, $p_2 \geq 0, \ldots$ が連立一次不等式
  p_i = `[`$p_{i,1}, p_{i,2}, \ldots, p_{i,k}$`]` のときは，$p_{i,1} \geq p_{i,2} \geq \cdots \geq p_{i,k}$ と解釈される
- E =`[`e_1`,`e_2`,...]` : $e_1 = 0$, $e_2 = 0, \ldots$ が連立される等式
  e の最後の成分または `E` が `[f,`e_k`]` で f が変数となっていれば，一次式 $f = e_k$ の値の範囲が返される（`[]` ならば無視）.
  以下の `[1]` では，$\frac{3}{4} \geq$ `m1` $\geq \frac{1}{2}$ が解で，`m1`$= \frac{5}{8}$ は解の例.
  `[7]` では，`f` = `m1` − `m2` のとる値の範囲が $\frac{1}{2} \geq$ `f` $\geq \frac{1}{6}$ となることを示す
- `var=`v : $v = [v_1, \ldots, v_n]$ で消去する変数の順序を指定
- `verb=1` : 各変数の範囲のデータが返される．最初のリストは変数の消去順序（cf. `[3]`）．もう 1 つのリストは最初が連立不等式のリスト（cf. `[4]`）で，以降は各変数の範囲を表す 3 成分のリストのリストで，その 3 成分（cf. `[5]`）は，第 1 成分の各式が非負のとき解があり，第 2 成分の各項以下で，第 3 成分の各項以上の範囲が解となる.
- `val=1` : 各変数の値が返される.
  解がない場合は，`[ ]` が返される（このときは，`val=1` でなくて `verb=1` を指定すれば状況が分かる）.
  以下の `[8]` は，`f` = `m1` + `2*m2` のとる値の範囲は $\frac{4}{3} \geq$ `f` ≥ 1 であって，(`m1`, `m2`) = $(\frac{2}{3}, \frac{1}{3})$ のとき `f` は値 $\frac{4}{3}$ を, $(\frac{1}{2}, \frac{1}{4})$ のときは 1 を取ること，さらに $(\frac{7}{12}, \frac{7}{24})$ のときは $\frac{7}{6}$ の値をとることを示している．変数名の最後に 1 か 0 が返され，条件の境界でなく内点に例が存在するかどうかを示す．`[9]` は，`g` = `2*m1` + `m2` の場合で，下図を参照.

$$\begin{cases} m_1 \geq 1 - m_1 \geq 0, \\ m_2 \geq 1 - 3m_2 \geq 0, \\ m_1 \geq m_2 \geq 1 - m_1 - m_2 \geq 0 \end{cases}$$

$\Rightarrow$

$$1 \leq m_1 + 2m_2 \leq \tfrac{4}{3}, \quad \tfrac{5}{4} \leq 2m_1 + m_2 \leq \tfrac{7}{4}.$$



実行例：

```
[0] P=[1-m1, [m1,1-m1],1-3*m2,[m2,1-3*m2],1-m1-m2,[m1,m2],[m2,1-m1-m2]]$
[1] os_md.linprog(P,[]);
[[m2,m1],5/8,3/4,1/2]
```

```
 [2] R=os_md.linprog(P,[]|verb=1)$
 [3] R[0];
 [m2,m1]
 [4] R[1][0];
 [[-m1+1,2*m1-1,-3*m2+1,4*m2-1,-m1-m2+1,m1-m2,m1+2*m2-1]]
 [5] R[1][1];
 [[1/12,1/2*m1-1/6,-m1+3/4,-1/2*m1+1/2,m1-1/4,3/2*m1-1/2,2*m1-1,-m1+1],
  [m1,-m1+1,1/3],[-1/2*m1+1/2,1/4]]
 [6] R[1][2];
 [[],[1,1,3/4],[1/2,1/3,1/4,1/3]]
 [7] os_md.linprog(P,[f,m1-m2]);
 [[m1,f],1/3,1/2,1/6]
 [8] os_md.linprog(P,[f,m1+2*m2]|val=1);
 [[f,m1,m2,1],[7/6,7/12,7/24],[4/3,2/3,1/3],[1,1/2,1/4]]]
 [9] os_md.linprog(P,[g,2*m1+m2]|val=1);
 [[g,m1,m2,1],[3/2,29/48,7/24],[7/4,3/4,1/4],[5/4,1/2,1/4]]
 [10] os_md.linprog([[m1,1-m1,0],[m2,1-3*m2,0],[m1,m2,1-m1-m2,0]],[]);
 [[m2,m1],5/8,3/4,1/2]
```

関数とそのアルゴリズム

```
// ------ 1 ------
def linprog(P,E)
{
  if(!isint(Verb=getopt(verb))) Verb=0;
  L=length(E);
  if(L==2&&isvar(car(E))&&type(E[1])!=4){
    L=1;E=[E];
  }
  if(!L||type(E[L-1])!=4||!os_md.isvar(V0=E[L-1][0])) V0=0;
  if(type(V=getopt(var))!=4){
    V=vars([P,E]);
    if(V0){
      V=lsort(V,[V0],"setminus");
      V=append(V,[V0]);
    }
  }
  for(E0=[];E!=[];E=cdr(E)){
    if(type(TE=car(E))==4) TE=TE[0]-TE[1];
    E0=cons(TE,E0);
  }
  E=reverse(E0);
// ------
```

零に等しい式のリスト E と変数（解く変数の優先順）のリスト V を定める．
最大値を調べる変数があれば，それを変数リストの最後に置く．

```
P=[1-m1, [m1,1-m1], 1-3*m2, [m2,1-3*m2], 1-m1-m2, [m1,m2], [m2,1-m1-m2]]
E=[g,2*m1+m2]
os_md.linprog(P,E|val=1);
 =>
L=[-m1+1,2*m1-1,-3*m2+1,4*m2-1,-m1-m2+1,m1-m2,m1+2*m2-1]
E=[g-2*m1-m2];
V=[m2,m1,g];
```

```
// ------ 2 ------
  S=lsol(E,V);
  for(L=[],TP=reverse(P);TP!=[];TP=cdr(TP)){
    if(type(Q=car(TP))==4)
      for(I=length(Q)-1;I>0;I--) L=cons(Q[I-1]-Q[I],L);
    else L=cons(Q,L);
  }
  for(SE=[],I=0;I<length(S);I++){
    L=subst(L,(S[I])[0],(S[I])[1]);
    SE=cons(S[I][0],SE);
  }
  SV=lsort(V,SE,"setminus");
// ------
```

連立一次方程式式 E の解 S （解いた変数と値のリスト）を求め非負とすべき式のリスト L に解を代入する．代入で消去された変数のリストを SE，残った変数のリストを SV とする．

```
S=[ [m2,g-2*m1] ]
L=[-m1+1,2*m1-1,-3*m2+1,4*m2-1,-m1-m2+1,m1-m2,m1+2*m2-1]
 =>  (m2 -> g-2*m1)
L=[-m1+1,2*m1-1,-3*g+6*m1+1,4*g-8*m1-1,-g+m1+1,-g+3*m1,2*g-3*m1-1]
SE=[m2]
SV=[m1,g]
```

```
// ------ 3 ------
  EQ=newvect((N=length(SV))+2);
  EQ[0]=[L];
  for(I=1;I<=N;I++){
    TV=SV[I-1];EQ[I]=[];
    for(R=R1=R2=[],T=EQ[I-1][0];T!=[];T=cdr(T)){
      C=coef(car(T),1,TV);
      if(C>0) R2=cons(-car(T)/C+TV,R2);
      else if(C<0) R1=cons(-car(T)/C+TV,R1);
      else EQ[I]=cons(car(T),EQ[I]);
    }
    if(I<N){
      for(T=R1;T!=[];T=cdr(T)){
        for(T1=R2;T1!=[];T1=cdr(T1))
          EQ[I]=cons(car(T)-car(T1),EQ[I]);
      }
    }
    EQ[I]=[EQ[I],R1,R2];
  }
// ------
```

連立一次不等式 L の変数を順に消去して連立不等式のベクトルリスト EQ を計算する．ベクトルのサイズは，変数 SV の個数（=:N）+2.

EQ[0] は，L

$i = 1, \ldots, N$ に対し，EQ[i] は，i 番目の変数 v_i までを消去した関係式のリストの 3 つからなる．それは EQ[$i-1$][0] にある非負となるべき各一次式から，i 番目の v_i の範囲を定める不等式として次のように求める．

一次式の v_i の係数が正なら，v_i が $v_{i+1}, v_{i+2}, \ldots$ のある一次式以上という式が得られ，その式の全体 R2 を EQ[i][2] に置く．一方，負の場合はある一次式以下とな

るので，その式の全体 R1 を EQ[i][1] に置く．係数が 0 なら，非負となるべき一次式なので，それが正数でなければ EQ[i][0] に置く．

さらに，$i < N$ のときは，R1 の各式が R2 の各式以上（差の一次式が非負），という式をすべて求めて EQ[i][0] に追加する．

不等式を満たす v_i の存在条件は，求められた EQ[i][0] のすべての一次式（変数は，$v_{i+1}, v_{i+2}, \ldots$）が非負となることであることに注意．

```
EQ[0]=[[-m1+1,2*m1-1,-3*g+6*m1+1,4*g-8*m1-1,-g+m1+1,-g+3*m1,2*g-3*m1-1]]
EQ[1]=[[-1/2*g+7/6,-g+2,-1/3*g+1,1/2*g-5/8,-1/2*g+7/8,1/6*g-1/8,2/3*g-5/6,
  1/6*g-1/6,-1/3*g+2/3,1/3*g-1/3],
 [2/3*g-1/3,1/2*g-1/8,1],[1/3*g,g-1,1/2*g-1/6,1/2]]
EQ[2]= [[], [2,7/4,3,2,7/3], [1,1,5/4,3/4,5/4]]
EQ[3]=EQ[N+1]=0
```

たとえば，EQ[1] の一項目は，2 項目と 3 項目の要素の差で，EQ[1][0] の最後の 1/3*g-1/3 は，(2/3*g-1/3)-(1/3+g)) に対応する．

```
// ------ 4 ------
  for(T=EQ[N][0];T!=[];T=cdr(T)){
    if((T0=car(T))==0) F=0;
    else if(type(T0)!=1) mycat(["Unknown condition", T0, "> 0"]);
    else if(T0<0){
         mycat(["Cannot resolve", T0,"> 0"]);
         return Verb?Eq:[];
  }
  U=lmin(R1);V=lmax(R2);F=1;
  if(U!=[]&&V!=[]){
    if(U<V){
      mycat(["No solution ",U,"<",V]);
      return Verb?EQ:[];
    }
    EQ[N+1]=[(U+V)/2,U,V];
  }else EQ[N+1]=[[],U,V];
// ------
```

EQ[N] から v_N の取り得る値の範囲を求める．それは EQ[N][1] の最小値以下で，EQ[N][2] の最大値以上．それらの中間値，v_N の最大値，v_N の最小値の 3 つの値のリストを EQ[N+1] とする．なお，lmax(), lmin() はリストの中の最大元，最小限を返す関数である．

ただし，v_N が存在しないときは [] を返す．

F=0 は不等号は不成立だが等号も許せば成り立つことを示す．

```
EQ[3]=[3/2,7/4,5/4]
```

```
// ------ 5 ------
  if(getopt(val)==1){
    R=[cons(SV[N-1],EQ[N+1])];
    RR=newvect(3);RS=newvect(3);
    for(J=0;J<3;J++) RR[J]=(car(R)[J+1]==[])?[]:EQ;
    for(J=0;J<3;J++) RS[J]=(car(R)[J+1]==[])?[]:S;
    for(I=N-1;I>0;I--){
      TR=car(R);
      for(F=1,SS=[],J=2;J>=0;J--){
        if(RR[J]==[]){
          SS=cons([],SS);continue;
        }
```

```
        RR[J]=subst(RR[J],SV[I],TR[J+1]);
        RS[J]=subst(RS[J],SV[I],TR[J+1]);
        U=RR[J][I];
        if(J==2) SS=cons(lmax(U[J]),SS);
        else if(J==1) SS=cons(lmin(U[J]),SS);
        else{
          U1=lmin(U[1]);U2=lmax(U[2]);
          if(U1==U2) F=0;
          SS=cons((U1+U2)/2,SS);
        }
      }
      R=cons(cons(SV[I-1],SS),R);
    }
// ------
```

v_N のみならず，v_N の値に対応する各変数 v_i の値を $i = N-1, N-2, \ldots$ の順に定めて，それを R に入れていく．.

R の各成分は，変数とその値 3 つのリストで，最初は v_N と EQ[N+1] とする．

RR および RS は，EQ および S に, 定めた変数 v_i の 3 種の値（TR=car(R) にある）を順に代入していったもの．

$v_i, \ldots, v_N$ の 3 種の値を EQ に代入した RR から v_{i-1} の 3 種の値を定めて，それを R に追加する．

```
TR= [g,3/2,7/4,5/4]
g -> 5/4 (最小値)：
RR[2]=[ [[-m1+1,2*m1-1,6*m1-11/4,-8*m1+4,m1-1/4,3*m1-5/4,-3*m1+3/2]]
 [[13/24,3/4,7/12,0,1/4,1/12,0,1/24,1/4,1/12],
  [1/2,1/2,1],[5/12,1/4,11/24,1/2]]
 [[], [2,7/4,3,2,7/3],[1,1,5/4,3/4,5/4]]
 [3/2,7/4,5/4] ]
v1=1/2  = max(RR[2][1][2]) = max([5/12,1/4,11/24,1/2])

g -> 7/4 (最大値)：
RR[1]= [ [[-m1+1,2*m1-1,6*m1-17/4,-8*m1+6,m1-3/4,3*m1-7/4,-3*m1+5/2]]
 [[7/24,1/4,5/12,1/4,0,1/6,1/3,1/8,1/12,1/4],
  [5/6,3/4,1],[7/12,3/4,17/24,1/2]]
 [[],[2,7/4,3,2,7/3],[1,1,5/4,3/4,5/4]]
 [3/2,7/4,5/4] ]
 v1=3/4 = min(RR[1][1][1]) = min([5/6,3/4,1])

g -> 3/2 (一般値)：
RR[0]= [ [[-m1+1,2*m1-1,6*m1-7/2,-8*m1+5,m1-1/2,3*m1-3/2,-3*m1+2]]
 [[5/12,1/2,1/2,1/8,1/8,1/8,1/6,1/12,1/6,1/6],
   [2/3,5/8,1],[1/2,1/2,7/12,1/2]]
 [[],[2,7/4,3,2,7/3],[1,1,5/4,3/4,5/4]]
 [3/2,7/4,5/4] ]
 v1=29/48 = (min(RR[0][1][1])+max(RR[0][1][2]))/2=(5/8+7/12)/2

R=[[m1,29/48,3/4,1/2], [g,3/2,7/4,5/4]]
RS=[ [ [m2,-2*m1+3/2] ] [ [m2,-2*m1+7/4] ] [ [m2,-2*m1+5/4] ] ]
```

```
// ------ 6 ------
    if(length(S)!=0){
      TR=car(R);
      for(I=length(S)-1;I>=0;I--){
```

```
        for(U=[],J=2;J>=0;J--)
          U=cons(subst(RS[J][I][1],TR[0],TR[J+1]),U);
        R=cons(cons(RS[0][I][0],U),R);
      }
    }
// ------
```

消去した各変数の値 U も求めて R に追加する．

RS[J] は S に $v_2,\ldots,v_N$ の 3 種の値を代入したもので，さらに v_1 の 3 種の値を TR=car(R) から得て代入すれば消去した変数の値が得られ，それを R に追加する．

```
RS <- (m1 -> 19/48, 3/4, 1/2)
 =>
R=[[m2,7/24,1/4,1/4], [m1,29/48,3/4,1/2], [g,3/2,7/4,5/4]]

// ------ 7 ------
    for(U=[],I=length(R[0])-1;I>=0;I--){
      for(TU=[],J=length(R)-1;J>=0;J--) TU=cons(R[J][I],TU);
      U=cons(TU,U);
    }
    R=cons(append(car(U),[F]),cdr(U));
    return Verb?[SV,EQ,R]:R;
  }
  return Verb?[SV,EQ]:cons(SV,EQ[N+1]);
}
// ------
```

各変数の値のリスト U を求める．

最後に，オプションに応じた値を返す．

```
U=[[g,m1,m2,1],[3/2,29/48,7/24],[7/4,3/4,1/4],[5/4,1/2,1/4]]
```

機能拡張 (2026.7.29)

無限小や無限大の正数を扱えるように機能拡張を行った．これは，多変数の複数の一次式が正や非負や 0 という条件の連立系に解があるのかどうかを調べるのが線型計画法のプログラムを書いた目的の 1 つであったからでもある．

無限大や無限小の正数は不定元のリスト，たとえば [ε_1,...,ε_k] で指定し，不定元でない数 1 がその中に 1 つだけ許される．たとえば $\varepsilon_\ell = 1$ ならば

$$\varepsilon_1 \gg \varepsilon_2 \gg \cdots \gg \varepsilon_{\ell-1} \gg \varepsilon_\ell = 1 \gg \varepsilon_{\ell+1} \gg \cdots \gg \varepsilon_k > 0$$

とみなす．ただし，1 がない場合は，リストの最後に 1 を追加したものが指定されたとみなす（無限大の正数のみ）．これらの変数の一次式で表された式を数とみなす．この式については，和や差や通常の数（有理数）との積は考えるが，(一次式でなくなるので）式同士の積は考えない．数とみなしたこの式は，linprog() の引数の一次不等式や等式の定数に用いることができる．すなわち，引数は通常の式の意味で一次式とする．

これは linprog() のオプションパラメータ ext で指定する：

- ext=[ε_1,...,ε_k]： 上のような無限大や無限小の正数を導入する．
- ext=1：引数で指定した等号つき不等号はすべて等号なしの不等号と解釈する．そのため，無限小の正数 e_ が自動的に導入される．

B=[ε_1,...,ε_k] としたとき，このような数 N の正負判定は，ispos(N,B) で，数のリストの最大元，最小元は lmaxx(L,B), lminx(L,B) で得られる．

linprog() のコードは lmax(*), lmin(*) を lmaxx(*,B), lmin(*,B) にすべて変更する他は以下の通り:

```
// ------ 1 ------ の部分の最後に追加
    if(type(B=getopt(ext))!=4) B=[];
    VB=vars(B);
    if(VB!=[]) V=lsort(V,VB,"setminus");

// ------ 2 ------ の部分に追加
    SV=lsort(V,SE,"setminus");
    if(B==[]&&getopt(ext)==1){
      B=[1,e_];VB=vars(B);
      L=calc(L,["-",e_]);
    }

// ------ 4 ------ の部分への追加・変更
    if((T0=car(T))==0) F=0;
      else if(type(T0)!=1){
        RB=lsort(vars(T0),VB,"setminus");
        if(RB!=[]) mycat(["Unknown condition", T0, "> 0"]);
      }
      else if(ispos(T0,B)<0){
        mycat(["Cannot resolve", T0,"> 0"]);
        if(!Verb) return [];
        return EQ;
      }
    }
    U=lminx(R1,B);V=lmaxx(R2,B);F=1;
    if(U!=[]&&V!=[]){
      if(ispos(V-U,B)>0){
        mycat(["No solution ",U,"<",V]);
        return Verb?EQ:[];
```

追加実行例：

```
[11] os_md.linprog(P,[]|ext=1,val=1);
[[m1,m2,1],[-3/8*e_+5/8,-1/24*e_+7/24],[-5/4*e_+3/4,1/4*e_+1/4],
 [1/2*e_+1/2,1/4*e_+1/4]]
```

Reference

[O] Oshima T., os_muldif.rr, a library of computer algebra Risa/Asir, 2008–2026.