УДК 519.85

АДАПТИВНЫЙ МЕТОД РЕШЕНИЯ ИНТЕРВАЛЬНОЙ ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ В СРЕДЕ MATLAB

Згонников А.В.
Санкт-Петербургский государственный университет, Санкт-Петербург,
e-mail: zzaarrkk@gmail.com

Смирнов Н.В.
Санкт-Петербургский государственный университет, Санкт-Петербург,
e-mail: nvs@apmath.spbu.ru

1. Введение. Как известно, линейное программирование зародилось в конце 30-х гг. прошлого века благодаря Л.В. Канторовичу. В работе [1] рассматриваются экономические модели, сводящиеся к задачам линейной оптимизации. В ней был предложен так называемый «метод разрешающих множителей», из которого впоследствии выросла теория двойственности.

Идеи Канторовича развил американский математик Дж. Данциг [2], разработавший в 1947 г. широко известный симплекс-метод, долгое время являвшийся единственным эффективным алгоритмом решения задач линейного программирования.

Однако, симплекс-метод имеет ряд существенных недостатков. Траектория его проходит через множество вершин выпуклого многогранника, образованного пересечениями областей, определяемых ограничениями задачи (см. рис.1).

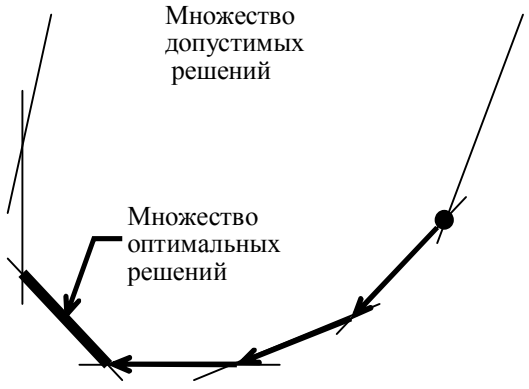

Рис. 1. Траектория симплекс-метода.

В то же время существует целый класс алгоритмов, не ограничивающихся перебором вершин, но допускающих прохождение траектории через внутренние точки множества допустимых значений – методы внутренних точек (см. рис.2). К таким методам относится разработанный в начале 1970-х гг. Р. Габасовым и Ф.М. Кирилловой адаптивный метод линейной оптимизации.

2. Адаптивный метод. Опишем кратко суть адаптивного метода, следуя работе [3].

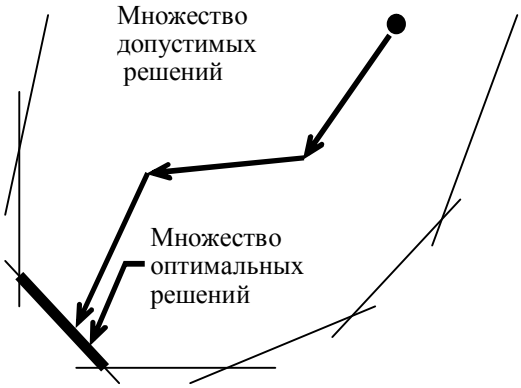

Рис. 2. Траектория методов внутренних точек.

Рассмотрим интервальную задачу линейного программирования (ИЗЛП):

$$\begin{cases} c^T x \to \max, \\ b_* \le Ax \le b^*, \\ d_* \le x \le d^*. \end{cases}$$

Здесь $c = (c_1,\ldots,c_n)^T$, $A_{m\times n}$, $x = (x_1,\ldots,x_n)^T$. Векторные неравенства следует понимать так: $a \ge b \Leftrightarrow \forall i : a_i \ge b_i$. Соотношения $b_* \le Ax \le b^*$ называются основными ограничениями, $d_* \le x \le d^*$ – прямыми. Вектор $z=Ax$ назовем вектором затрат.

Введем множества $I = \{1,\ldots,m\}$ и $J = \{1,\ldots,n\}$ индексов строк и столбцов матрицы основных ограничений. Совокупность двух подмножеств $I_{оп} \subseteq I$ и $J_{оп} \subseteq J$ назовем опорой $K_{оп} = \{I_{оп}, J_{оп}\}$, если $|I_{оп}| = |J_{оп}|$ и $\det A(I_{оп}, J_{оп}) \ne 0$.

Пара $\{x, K_{оп}\}$ называется опорным планом. Опорный план называется невырожденным, если $b_{*н} < z_н < b^*_н$ и $d_{*оп} < x_{оп} < d^*_{оп}$, то есть неопорные компоненты вектора затрат и опорные компоненты плана не выходят на границу множества допустимых решений.

Легко убедиться в том, что если $\{x, K_{оп}\}$ – опорный план, то $x_{оп}$ и $z_н$ можно однозначно найти, если задать $x_н$ и $z_{оп}$, то есть $x_{оп}$ и $z_н$ – зависимые переменные. Это позволяет выразить произвольное приращение целевой функции $c^T x$ через приращения $x_н$ и $z_{оп}$.

Каждой опоре в задаче соответствуют два вектора – вектор потенциалов *u(I)* и вектор оценок $\Delta(J)$, определяемые равенствами:

$$\begin{cases} u(I_{оп}) = (A(I_{оп}, J_{оп})^{-1})^T c(J_{оп}), \\ u(I_н) = 0; \end{cases}$$

$$\begin{cases} \Delta(J_{оп}) = 0, \\ \Delta(J_н) = c(J_н) - A(I_{оп}, J_н)^T u(I_{оп}). \end{cases}$$

Физический смысл этих векторов состоит в том, что они являются условными градиентами целевой функции при определенных, зависящих от опоры изменениях векторов *z* и *x*.

Справедливо следующее утверждение.

Теорема 1 [3]. *Для оптимальности плана x достаточно существования опоры* $K_{оп}$, *при которой на опорном плане* $\{x, K_{оп}\}$ *выполняются соотношения:*

$$\begin{cases} u_i \le 0, \quad z_i = b_{*i}, \\ u_i \ge 0, \quad z_i = b_i^*, \\ u_i = 0, \ b_{*i} < z_i < b_i^*, \\ i \in I_{ОП}; \end{cases} \tag{1}$$

$$\begin{cases} \Delta_j \le 0, x_j = d_{*j}, \\ \Delta_j \ge 0, x_j = d_j^*, \qquad j \in J_H, \\ \Delta_j = 0, d_{*j} < x_j < d_i^*. \end{cases} \tag{2}$$

Обратно, если x – оптимальный план, и при некоторой опоре $K_{оп}$ *пара* $\{x, K_{оп}\}$ *– невырожденный опорный план, то на этом опорном плане выполняются соотношения* (1), (2).

Данный критерий оптимальности становится интуитивно понятен, если учесть, что u_i и Δ_j – условные скорости изменения целевой функции: знаки скоростей должны быть такими, чтобы допустимые изменения независимых переменных не могли привести к увеличению значения целевой функции.

Пусть задана опора $K_{оп}$ и соответствующие ей векторы оценок и потенциалов. Построенные исходя из условий оптимальности векторы χ и ξ ($\chi_{оп}$ и $\xi_н$ выражаются через $\chi_н$ и $\xi_{оп}$) назовем, соответственно, сопровождающим псевдопланом и сопровождающим вектором псевдозатрат. Нетрудно заметить, что если полученные векторы удовлетворяют всем ограничениям нашей задачи, то вектор χ является оптимальным планом.

Часто при решении прикладных задач не требуется нахождения точного максимума целевой функции, а достаточно лишь приближенного значения оптимального плана. План x^ε назовем ε-оптимальным, если значение целевой функции на нем отличается от значения на оптимальном плане не более чем на ε. Адаптивный метод решения ИЗЛП, в отличие от симплекс-метода, позволяет по заданному ε находить ε-оптимальный план.

Число $\beta(x, K_{оп}) = c^T \chi - c^T x$ назовем оценкой субоптимальности опорного плана $\{x, K_{оп}\}$. Если $\bar{x}$ – оптимальный план, то существует такая опора $\overline{K}_{оп}$, что $\beta(\bar{x}, \overline{K}_{оп}) = 0$. Можно сформулировать критерий ε-оптимальности.

Теорема 2 [3]. *Для любого* ε *для* ε*-оптимальности плана x необходимо и достаточно существования опоры* $K_{оп}$ *такой, что*

$\beta(x, K_{оп}) \le \varepsilon$.

При решении реальных задач может быть известно, что с большой долей вероятности некоторая компонента плана примет определенное значение, либо что некоторое ограничение, скорее всего, будет критическим. Адаптивный метод позволяет учитывать имеющуюся информацию о решении, сокращая, таким образом, объем вычислений.

Адаптивный метод является итеративным и состоит из двух фаз. Первая фаза – построение начального плана, вторая – построение последующих приближений.

Вообще, построение какого-либо плана ИЗЛП – нетривиальная задача, в адаптивном методе она решается с помощью применения алгоритма второй фазы к вспомогательной ИЗЛП большей размерности с известным начальным планом.

Алгоритм второй фазы заключается в последовательном применении к начальному опорному плану двух процедур – замены плана и замены опоры. Замена плана происходит так: из начальной точки x движемся в направлении сопровождающего псевдоплана χ. Это направление наибольшего возрастания функции по неопорным компонентам плана и опорным компонентам вектора затрат. Движение происходит до нарушения опорных прямых ограничений или неопорных основных (другие ограничения не могут нарушиться по построению сопровождающего псевдоплана). Если нарушения не произошло, то мы попадем в точку χ. Она и будет оптимальным планом.

Правило замены опоры зависит от того, какое именно ограничение стало активным на этапе замены плана. Подробное описание процедур замены плана и опоры можно найти в работе [3].

Подытожив все вышеизложенное, сравним адаптивный метод с самым распространенным на данный момент симплекс-методом:

1. Адаптивный метод находит ε-приближенное (причем ε можно положить равным нулю) решение задачи, что, во-первых, позволяет в некоторых прикладных задачах серьезно сэкономить на вычислениях, а во-вторых, избавляет от необходимости оценивать погрешность вычислений.
2. Классический симплекс-метод требует сведения задачи линейного программирования к канонической форме с односторонними прямыми ограничениями:

$$\begin{cases} c^T x \to \max, \\ Ax = b, \\ x \geq 0. \end{cases}$$

   Как нетрудно заметить, каноническая задача является частным случаем ИЗЛП. В то же время сведение интервальной задачи к канонической (см. [3]) требует многократного увеличения размерности, что отрицательно сказывается на эффективности метода. С этой точки зрения ИЗЛП более предпочтительна, а, следовательно, и адаптивный метод – эффективнее.
3. Как уже было замечено, адаптивный метод относится к методам внутренних точек. Как правило, такие методы позволяют найти оптимальное решение за меньшее число итераций. Кроме этого, найти вершину многогранного множества большой размерности – довольно сложная задача, особенно для не математика. Адаптивный метод построен так, что любое предложение специалистов (даже не являющееся планом) может быть принято в качестве начального приближения.
4. В симплекс-методе аналогом опоры является базис, причем он однозначно соответствует базисному решению. В адаптивном методе опора совершенно не зависит от плана, что позволяет менять их независимо друг от друга и более эффективно строить оптимальный план.
5. Благодаря гибкости опоры, адаптивный метод хорошо работает с задачами, в которых количество переменных достаточно велико относительно числа основных ограничений (такие задачи часто и встречаются в приложениях).

Изложенные идеи адаптивного метода имеют перспективы не только в задачах линейного программирования, но и при построении оптимальных управлений в

линейных и нелинейных управляемых системах. Более того, адаптивный метод можно применять при решении задач позиционной оптимизации. Соответственно, написанная программа является первым блоком более широкого комплекса программных средств, позволяющего рассчитывать оптимальные и стабилизирующие управления для различных классов динамических систем в режиме реального времени.

Среда Matlab была выбрана для реализации рассматриваемого метода главным образом из-за своей ориентированности на работу с векторами и матрицами. Преимущества языка Matlab, такие как логическая индексация массивов, отсутствие необходимости самостоятельно реализовывать перемножение векторов и матриц, сделали работу с инструментами метода (главным образом, с опорой) максимально удобной и простой. Блок-схема программы показана на рис.3.

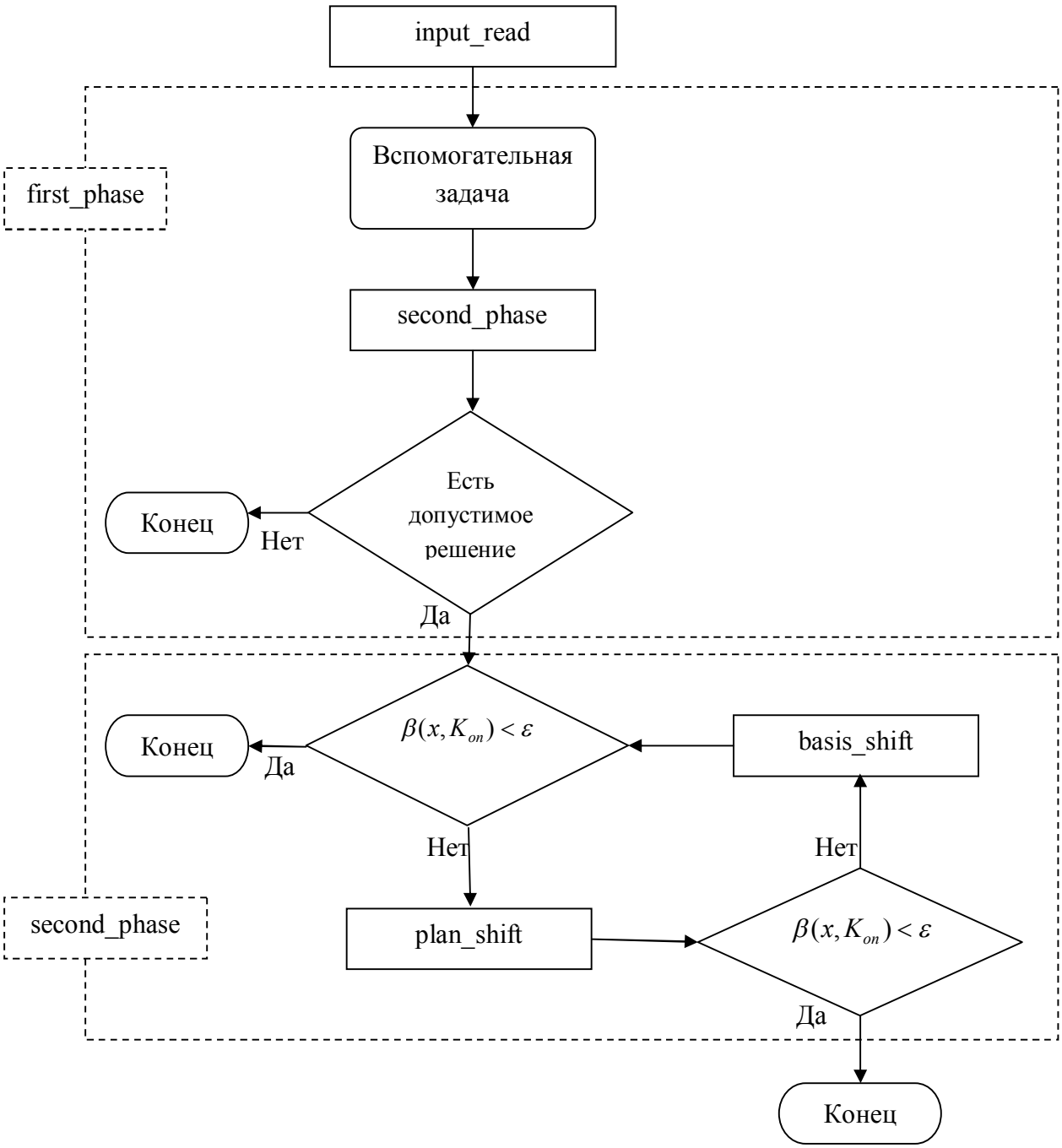

Рис. 3. Блок-схема программы.

3. Код программы. В данном разделе приводится полный текст программы, реализующей адаптивный метод решения ИЗЛП.

Главная процедура

```
clear;
input_read;
[x,bas_I,bas_J,pl_cor_ind]=first_phase(A,b_l,b_u,d_l,d_u);
if(pl_cor_ind)
   [x_e,bas_I_e,bas_J_e]=second_phase(x,bas_I,bas_J,e,c,A,b_l,b_u,d_l,d_u)
else
  display ('Нет допустимых планов');
end
```

Процедура“input_read”

```
%параметры задачи читаются из файла input.txt
fid = fopen('input.txt');
m = str2double(fgetl(fid));
n = str2double(fgetl(fid));
e = str2double(fgetl(fid));
c = (str2num(fgetl(fid)))';
for i=1:m
   A(i,:)=str2num(fgetl(fid));
end
b_l = (str2num(fgetl(fid)))';
b_u = (str2num(fgetl(fid)))';
d_l = (str2num(fgetl(fid)))';
d_u = (str2num(fgetl(fid)))';
```

Процедура “first_phase”

```
function [x,bas_I,bas_J,pl_cor_ind]=first_phase(A,b_l,b_u,d_l,d_u)
%первая фаза адаптивного метода
[m,n]=size(A);
pl_cor_ind = true;
%индикатор корректности задачи
%по умолчанию считаем, что задача корректна
x=(d_l+d_u)/2;
bas_I=[];
bas_J=[];
%отправная точка всех вычислений
%вместо центра множества, определяемого прямыми ограничениями
%и пустой опоры можно использовать имеющуюся информацию о плане и опоре
z=A*x;
for i=1:m
   if (z(i)<b_l(i))
      z(i)=b_l(i);
   else if(z(i)>b_u(i))
         z(i)=b_u(i);
      end
   end
end
```

```
w=z-A*x;
%вектор невязок
%если он равен нулю, то начальная точка является планом задачи
temp = false;
for i=1:m
   if(w(i)~=0)
      temp = true;
   end
end
%если же нет, то решаем вспомогательную задачу
if(temp)
   q=zeros(n+m,1);
   K=zeros(m,m);
   for i=1:m
      if(w(i)<0)
         K(i,i)=-1;
         q(n+i)=-1;
      else if (w(i)>0)
            K(i,i)=1;
            q(n+i)=-1;
         end
      end
   end
   d_l=[d_l;zeros(m,1)];
   d_u=[d_u;abs(w)];
   x=[x;abs(w)];
   A=[A,K];
   [x,bas_I,bas_J]=second_phase(x,bas_I,bas_J,0,q,A,b_l,b_u,d_l,d_u);
   %проверка полученного решением вспомогательной задачи вектора
   %если последние т компонент не равны нулю, то у исходной задачи нет
   %допустимых решений
   for i=1:m
      if(x(n+i))
         pl_cor_ind = false;
      end
   end
   x((n+1):(n+m))=[];
   %если оптимальная для вспомогательной задачи опора не содержит индексов
   %искусственных переменных, то можно взять ее в качестве начальной для
   %исходной задачи
   if(pl_cor_ind)
      bas_cor_ind = true;
      for i=1:m
         if(ismember(n+i,bas_I)||ismember(n+i,bas_J))
            bas_cor_ind = false;
         end
      end
      if(~bas_cor_ind)
         bas_I = [];
         bas_J = [];
```

```
        end
    end
end
```

Процедура “second _phase”

```
function [x_e,bas_I_e,bas_J_e]=second_phase(x,bas_I,bas_J,e,c,A,b_l,b_u,d_l,d_u)
%вторая фаза адаптивного метода
while (b(x,bas_I,bas_J,c,A,b_l,b_u,d_l,d_u)>e)
    [x,number,direct_ind]=plan_shift(x,bas_I,bas_J,e,c,A,b_l,b_u,d_l,d_u);
    if(number~=0)
        if(b(x,bas_I,bas_J,c,A,b_l,b_u,d_l,d_u)>e)
          [bas_I,bas_J]=basis_shift(x,c,A,b_l,b_u,d_l,d_u,number,direct_ind,bas_I,bas_J);
        end
    else
        break
    end
end
x_e = x;
bas_I_e=bas_I;
bas_J_e=bas_J;
```

Функция “plan_shift”

```
function [y,number,direct_ind] = plan_shift(x,bas_I,bas_J,e,c,A,b_l,b_u,d_l,d_u)
%функция замены плана
%y - новый план,
%number - номер активного ограничения
%(0, если план оптимален или e-оптимален),
%direct_ind - "true", если
%активное ограничение - прямое, "false" - если основное
[m,n]=size(A);
z=A*x;
cpp=comp_v(bas_I,bas_J,A,c,b_l,b_u,d_l,d_u);
%l-направление наибольшего возрастания функции
l=cpp-x;
p=A*l;
th_x=zeros(n,1);
th_z=zeros(m,1);
%вычисление оптимального шага
for j=1:n
    th_x(j) = inf;
    if(ismember(j,bas_J))
        if(l(j)>0)
            th_x(j)=(d_u(j)-x(j))/l(j);
        else if(l(j)<0)
                th_x(j)=(d_l(j)-x(j))/l(j);
            end
        end
    end
end
for i=1:m
```

```
    th_z(i)=inf;
    if(~ismember(i,bas_I))
       if(p(i)>0)
          th_z(i)=(b_u(i)-z(i))/p(i);
       else if(p(i)<0)
             th_z(i)=(b_l(i)-z(i))/p(i);
          end
       end
    end
 end
 [th_x_0,j_0]=min(th_x);
 [th_z_0,i_0]=min(th_z);
 [th_0,k_0]=min([1, th_x_0, th_z_0]);
 y=x+th_0*l;
 direct_ind=false;
 number=0;
 if(c'*(cpp-y)>=e)
    if (k_0==2)
       number=j_0;
       direct_ind = true;
    else if (k_0==3)
          number=i_0;
       end
    end
 end
```

Функция “basis_shift”

```
 function [m_bas_I, m_bas_J]=
 basis_shift(x,c,A,b_l,b_u,d_l,d_u,number,direct_ind,bas_I,bas_J)
 %функция замены опоры
 %параметр direct_ind показывает, какое ограничение нарушилось
 %на этапе замены плана - равен true, если прямое, false - если основное
 [m,n]=size(A);
 z=A*x;
 [u,delta]=grad_v(bas_I,bas_J,c,A);
 [cpp,cev,degenerate_case]=comp_v(bas_I,bas_J,A,c,b_l,b_u,d_l,d_u);
 non_bas_J = non_(bas_J,n);
 A_b=A(bas_I,bas_J);
 A_b_nb=A(bas_I,non_bas_J);
 dy=zeros(m,1);
 dd=zeros(n,1);
 if(~isempty(bas_I))
    %случай непустой опоры
    if (direct_ind)
       g=cpp(number)-x(number);
       dd(number)=sign(g);
       dy(bas_I)=-inv(A_b)'*dd(bas_J);
       dd(non_bas_J) = -A_b_nb'*dy(bas_I);
    else
       g=cev(number)-z(number);
```

```
    dy(number)=sign(g);
    dy(bas_I)=-sign(g)*inv(A_b)'*A(number,bas_J)';
    dd(non_bas_J)=-dy'*A(:,non_bas_J);
end
alpha = -abs(g);
s_i = zeros(m,1);
s_j = zeros(n,1);
if(degenerate_case)
    %в случае двойственно вырожденного опорного плана возможны два
    %варианта: alpha <0 и alpha>=0
    %в первом случае действуем так же, как в невырожденном случае
    I_0 = bas_I(u(bas_I)==0);
    J_0 = non_bas_J(delta(non_bas_J)==0);
    alpha=alpha+sum((b_u(I_0)-b_l(I_0))'*dy(I_0))+sum((d_u(J_0)-
    d_l(J_0))'*dd(J_0));
end
if(~(alpha<0)&&degenerate_case)
    %если же alpha>=0 то берем произвольный индекс из I_0 или J_0
    if (~isempty(I_0))
        k_0 = 1;
        i_ = I_0(1);
    else
        k_0 = 2;
        j_ = J_0(1);
    end
else
    %вычисление короткого двойственного шага в невырожденном случае и
    %при непустой опоре
    for i=1:m
        s_i(i)=inf;
        if(ismember(i,bas_I))
            if(u(i)*dy(i)<0)
                s_i(i)=-u(i)/dy(i);
            end
        end
    end
    for j=1:n
        s_j(j)=inf;
        if(~ismember(j,bas_J))
            if(delta(j)*dd(j)<0)
                s_j(j)=-delta(j)/dd(j);
            end
        end
    end
    [s_i_0,i_]=min(s_i);
    [s_j_0,j_]=min(s_j);
    [s_0,k_0]=min([s_i_0,s_j_0]);
end
%k_0 показывает, компонента какой из двойственных переменных
%(1, если вектора Лагранжа, 2, если коплана) первой меняет знак
```

```
    %при коротком двойственном шаге
    if((~direct_ind)&&(k_0==1))
       bas_I(bas_I==i_)=number;
       m_bas_I = sort(bas_I);
       m_bas_J = bas_J;
    end
    if((~direct_ind)&&(k_0==2))
       bas_I = [bas_I,number];
       bas_J = [bas_J,j_];
       m_bas_I = sort(bas_I);
       m_bas_J = sort(bas_J);
    end
    if((direct_ind)&&(k_0==1))
       bas_I(bas_I==i_)=[];
       bas_J(bas_J==number)=[];
       m_bas_I = bas_I;
       m_bas_J = bas_J;
    end
    if((direct_ind)&&(k_0==2))
       bas_J(bas_J==number)=j_;
       m_bas_I = bas_I;
       m_bas_J = sort(bas_J);
    end
 else
    %случай пустой опоры
    g = cev(number)-z(number);
    dy(number)=sign(g);
    dd=-A'*dy;
    alpha=-abs(g);

    if(degenerate_case)
       J_0 = non_bas_J(delta(non_bas_J)==0);
       alpha=alpha+sum((d_u(J_0)-d_l(J_0))'*dd(J_0));
    end
    if(~(alpha<0)&&degenerate_case)
       j_ = J_0(1);
    else
       for j=1:n
          s_j(j)=inf;
          if(delta(j)*dd(j)<0)
             s_j(j)=-delta(j)/dd(j);
          end
       end
       [s_j_0,j_]=min(s_j);
    end
    m_bas_I = [bas_I,number];
    m_bas_J = [bas_J,j_];
 end
```

Функция “b”

```
function y = b(x,bas_I,bas_J,c,A,b_l,b_u,d_l,d_u)
%вычисление оценки субоптимальности
cpp=comp_v(bas_I,bas_J,A,c,b_l,b_u,d_l,d_u);
y=c'*(cpp-x);
```

Функция "comp_v"

```
function [cpp,cev,degenerate_case] = comp_v(bas_I,bas_J,A,c,b_l,b_u,d_l,d_u)
%comp_v - companion vectors - сопровождающие векторы
%cpp - companion pseudo plan - сопровождающий псевдоплан
%cev - companion pseudo expense vector - сопровождающий вектор псевдозатрат
[m,n]=size(A);
non_bas_I=non_(bas_I,m);
non_bas_J=non_(bas_J,n);
A_b = A(bas_I,bas_J);
A_b_nb = A(bas_I,non_bas_J);
[u,delta] = grad_v(bas_I,bas_J,c,A);
cpp=zeros(n,1);
cev=zeros(m,1);
degenerate_case=false;
if(ismember(0,u(bas_I))||ismember(0,delta(non_bas_J)))
   %случай двойственно вырожденного опорного плана
   degenerate_case=true;
   I_0 = bas_I(u(bas_I)==0);
   J_0 = non_bas_J(delta(non_bas_J)==0);
   if(~isempty(I_0))
      I_0_p = I_0(1);
      I_0(1) = [];
      I_0_m = I_0;
      cev(I_0_m)=b_l(I_0_m);
      cev(I_0_p)=b_u(I_0_p);
   end
   if(~isempty(J_0))
      J_0_p = J_0(1);
      J_0(1) = [];
      J_0_m = J_0;
      cpp(J_0_m)=d_l(J_0_m);
      cpp(J_0_p)=d_u(J_0_p);
   end
   I_p = bas_I(u(bas_I)>0);
   I_m = bas_I(u(bas_I)<0);
   J_p = non_bas_J(delta(non_bas_J)>0);
   J_m = non_bas_J(delta(non_bas_J)<0);
   cev(I_m)=b_l(I_m);
   cev(I_p)=b_u(I_p);
   cpp(J_m)=d_l(J_m);
   cpp(J_p)=d_u(J_p);
else
   cev(bas_I)=(b_l(bas_I)+b_u(bas_I))/2;
   cev(bas_I(u(bas_I)<0))=b_l(bas_I(u(bas_I)<0));
   cev(bas_I(u(bas_I)>0))=b_u(bas_I(u(bas_I)>0));
```

```
    cpp(non_bas_J)=(d_l(non_bas_J)+d_u(non_bas_J))/2;
    cpp(non_bas_J(delta(non_bas_J)<0))=d_l(non_bas_J(delta(non_bas_J)<0));
    cpp(non_bas_J(delta(non_bas_J)>0))=d_u(non_bas_J(delta(non_bas_J)>0));
    if (~isempty(bas_I))
       cpp(bas_J) = inv(A_b)*(cev(bas_I) - A_b_nb*cpp(non_bas_J));
    end
    cev(non_bas_I)=A(non_bas_I,:)*cpp;
  end
```

Функция "grad_v"

```
 function [u,delta] = grad_v(bas_I,bas_J,c,A)
 %функция подсчета векторов потенциалов и оценок
 [m,n]=size(A);
 non_bas_J=non_(bas_J,n);
 A_b=A(bas_I,bas_J);
 A_b_nb=A(bas_I,non_bas_J);
 u = zeros(m,1);
 delta = zeros(n,1);
 if(isempty(bas_I))
    delta = c;
 else if(isempty(non_bas_J))
       u = inv(A_b)'*c;
    else
       u(bas_I) = inv(A_b)'*c(bas_J);
       delta(non_bas_J) = c(non_bas_J)-A_b_nb'*u(bas_I);
    end
 end
```

Функция "non_"

```
 function non_bas = non_(bas,n)
 %заполнение массива неопорных индексов
 % (чисел от 1 до n, не входящих в вектор bas)
 non_bas=[];
 for j=1:n
    if(~ismember(j,bas))
       non_bas = [non_bas,j];
    end
 end
```

Структура входного файла

m –количество основных ограничений
n – количество переменных
e – точность
c – вектор целевой функции
A – матрица затрат
b_l – нижняя граница вектора затрат
b_u – верхняя граница вектора затрат
d_l – нижняя граница ограничений
d_u – верхняя граница ограничений

Литература

1. *Канторович Л.В.* Математические методы организации и планирования производства. Л.: ЛГУ, 1939, 68 с.
2. *Данциг Дж.* Линейное программирование, его применения и обобщения. М.: Прогресс, 1966, 600 с.
3. *Альсевич В.В., Габасов Р., Глушенков В.С.* Оптимизация линейных экономических моделей. Статические задачи. Мн.: БГУ, 2000, 210 с.