Tutorial 6

Linear Programming with MATLAB

Math 472/CS 472

Edward Neuman
Department of Mathematics
Southern Illinois University at Carbondale
edneuman@siu.edu

This tutorial is devoted to the discussion of computational tools that are of interest in linear programming (LP). MATLAB powerful tools for computations with vectors and matrices make this package well suited for solving typical problems of linear programming. Topics discussed in this tutorial include the basic feasible solutions, extreme points and extreme directions of the constraint set, geometric solution of the linear programming problem, the Two-Phase Method, the Dual Simplex Algorithm, addition of a constraint and Gomory's cutting plane algorithm.

6.1 MATLAB functions used in Tutorial 6

Function	Description
abs	Absolute value
all	True if all elements of a vector are nonzero
any	True if any element of a vector is nonzero
axis	Control axis scaling and appearance
break	Terminate execution of for or while loop
clc	Clear Command Window
convhull	Convex hull
diff	Difference and approximate derivative
disp	Display array
eps	Floating point relative accuracy
eye	Identity matrix
find	Find indices of nonzero of nonzero elements
FontSize	Size of a font
gca	Get handle to current axis
get	Get object properties
grid	Grid lines
hold	Hold current graph
inf	Infinity

intersect	Set intersection
isempty	True for empty matrix
length	Length of vector
LineStyle	Style of a line
LineWidth	Width of a line
max	Largest component
min	Smallest component
msgbox	Message box
nchoosek	Binomial coefficient or all combinations
patch	Create patch
pause	Wait for user response
plot	Linear plot
questdlg	Question dialog box
return	Return to invoking function
set	Set object properties
size	Size of matrix
sprintf	Write formatted data to string
sqrt	Square root
strcmp	Compare strings
sum	Sum of elements
title	Graph title
union	Set union
varargin	Variable length input argument list
varargout	Variable length output argument list
warning off	Suppresses all subsequent warning messages
xlabel	X-axis label
ylabel	Y-axis label
zeros	Zeros array

To learn more about a particular MATLAB function type **help *functionname*** in the **Command Window** and next press the **Enter** key.

6.2 Notation

The following symbols will be used throughout the sequel.

- $\mathbb{R}^n$ – n-dimensional Euclidean vector space. Each member of this space is an n-dimensional column vector. Lower case letters will denote members of this space.
- $\mathbb{R}^{m \times n}$ – collection of all real matrices with m rows and n columns. Upper case letters will denote members of this space.
- $\mathbf{T}$ – operator of transposition. In MATLAB the *single quote operator* **'** is used to create transposition of a real vector or a real matrix.
- $\mathbf{x}^T\mathbf{y}$ – the inner product (dot product) of $\mathbf{x}$ and $\mathbf{y}$.
- $\mathbf{x} \geqslant \mathbf{0}$ – nonnegative vector. All components of $\mathbf{x}$ are greater than or equal to zero.

6.3 Five auxiliary MATLAB functions

Some MATLAB functions that are presented in the subsequent sections of this tutorial make calls to functions named **ver**, **delcols**, **MRT**, **MRTD** and **Br**. These functions should be saved in the directory holding other m-files that are used in this tutorial.

```
function e = vr(m,i)

% The ith coordinate vector e in the m-dimensional Euclidean space.

e = zeros(m,1);
e(i) = 1;
```

```
function d = delcols(d)

% Delete duplicated columns of the matrix d.

d = union(d',d','rows')';
n = size(d,2);
j = [];
for k =1:n
   c = d(:,k);
   for l=k+1:n
      if norm(c - d(:,l),'inf') <= 100*eps
         j = [j l];
      end
   end
end
if ~isempty(j)
   j = sort(j);
   d(:,j) = [ ];
end
```

First line of code in the body of function **delcols** is borrowed from the MATLAB's help file. Two vectors are regarded as duplicated if the corresponding entries differ by more than 100 times the *machine epsilon*. In MATLAB this number is denoted by **eps** and is approximately equal to

```
format long

eps

ans =
    2.220446049250313e-016

format short
```

In order to display more digits we have changed the default format (**short**) to **long**. To learn more about available formats in MATLAB type **help format** in the **Command Window**.

```
function [row, mi] = MRT(a, b)

% The Minimum Ratio Test (MRT) performed on vectors a and b.
% Output parameters:
% row – index of the pivot row
% mi – value of the smallest ratio.

m = length(a);
c = 1:m;
a = a(:);
b = b(:);
l = c(b > 0);
[mi, row] = min(a(l)./b(l));
row = l(row);



function col = MRTD(a, b)

% The Maximum Ratio Test performed on vectors a and b.
% This function is called from within the function dsimplex.
% Output parameter:
% col - index of the pivot column.

m = length(a);
c = 1:m;
a = a(:);
b = b(:);
l = c(b < 0);
[mi, col] = max(a(l)./b(l));
col = l(col);



function [m, j] = Br(d)

% Implementation of the Bland's rule applied to the array d.
% This function is called from within the following functions:
% simplex2p, dsimplex, addconstr, simplex and cpa.
% Output parameters:
% m - first negative number in the array d
% j - index of the entry m.

ind = find(d < 0);
if ~isempty(ind)
   j = ind(1);
   m = d(j);
else
   m = [];
   j = [];
end
```

6.4 Basic feasible solutions

The *standard form* of the linear programming problem is formulated as follows. Given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and two vectors $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{b} \geqslant \mathbf{0}$, find a vector $\mathbf{x} \in \mathbb{R}^n$ such that

$$\begin{aligned} \textbf{min} \quad & z = \mathbf{c}^T\mathbf{x} \\ \textbf{Subject to} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geqslant \mathbf{0} \end{aligned}$$

Function **vert = feassol(A, b)** computes all basic feasible solutions, if any, to the system of constraints in standard form.

```
function vert = feassol(A, b)

% Basic feasible solutions vert to the system of constraints

%                       Ax = b, x >= 0.

% They are stored in columns of the matrix vert.

[m, n] = size(A);
warning off
b = b(:);
vert = [];
if (n >= m)
   t = nchoosek(1:n,m);
   nv = nchoosek(n,m);
   for i=1:nv
      y = zeros(n,1);
      x = A(:,t(i,:))\b;
      if all(x >= 0 & (x ~= inf & x ~= -inf))
         y(t(i,:)) = x;
         vert = [vert y];
      end
   end
else
   error('Number of equations is greater than the number of
variables.')
end
if ~isempty(vert)
   vert = delcols(vert);
else
   vert = [];
end
```

To illustrate functionality of this code consider the following system of constraints (see [1], Example 3.2, pp. 85-87):

$$\begin{aligned} x_1 + x_2 &\leqslant 6 \\ x_2 &\leqslant 3 \end{aligned}$$

$$x_1, x_2 \geq 0.$$

To put this system in standard form two *slack variables* x_3 and x_4 are added. The constraint matrix **A** and the right hand sides **b** are

```
A = [1 1 1 0; 0 1 0 1];

b = [6; 3];

vert = feassol(A, b)

vert =
     0     0     3     6
     0     3     3     0
     6     3     0     0
     3     0     0     3
```

To obtain values of the legitimate variables x_1 and x_2 it suffices to extract rows one and two of the matrix **vert**

```
vert = vert(1:2, :)

vert =
     0     0     3     6
     0     3     3     0
```

6.5 Extreme points and extreme directions of the constraint set

Problem discussed in this section is formulated as follows. Given a polyhedral set $X = \{x: Ax \leq b$ or $Ax \geq b$, $x \geq 0\}$ find all *extreme points* **t** of **X**. If **X** is *unbounded*, then in addition to finding the extreme points **t** its *extreme directions* **d** should be determined as well. To this end we will assume that the constraint set does not involve the equality constraints. If the LP problem has the equality constraint, then one can replace it by two inequality constraints. This is based on the following trivial observation: the equality $a = b$ is equivalent to the system of inequalities $a \leq b$ and $a \geq b$. Knowledge of the extreme points and extreme directions of the polyhedral set **X** is critical for a full mathematical description of this set. If set **X** is bounded, than a *convex combination* of its extreme points gives a point in this set. For the unbounded sets, however, a convex combination of its extreme points and a *linear combination*, with positive coefficients, of its extreme directions gives a point in set **X**. For more details, the interested reader is referred to [1], Chapter 2.

Extreme points of the set in question can be computed using function **extrpts**

```
function vert = extrpts(A, rel, b)

% Extreme points vert of the polyhedral set

%               X = {x: Ax <= b or Ax >= b, x >= 0}.

% Inequality signs are stored in the string rel, e.g.,
```

```
% rel = '<<>' stands for <= , <= , and >= , respectively.

[m, n] = size(A);
nlv = n;
for i=1:m
   if(rel(i) == '>')
      A = [A -vr(m,i)];
   else
      A = [A vr(m,i)];
   end
   if b(i) < 0
      A(i,:) = - A(i,:);
      b(i) = -b(i);
   end
end
warning off
[m, n]= size(A);
b = b(:);
vert = [ ];
if (n >= m)
   t = nchoosek(1:n,m);
   nv = nchoosek(n,m);
   for i=1:nv
      y = zeros(n,1);
      x = A(:,t(i,:))\b;
      if all(x >= 0 & (x ~= inf & x ~= -inf))
         y(t(i,:)) = x;
         vert = [vert y];
      end
   end
else
   error('Number of equations is greater than the number of variables')
end
vert = delcols(vert);
vert = vert(1:nlv,:);
```

Consider the polyhedral set **X** given by the inequalities

$$-x_1 + x_2 \leq 1$$
$$-0.1x_1 + x_2 \leq 2$$
$$x_1, x_2 \geq 0$$

To compute its extreme points we define

```
A = [-1 1; -0.1 1];

rel = '<<';

b = [1; 2];
```

and next run the m-file **extrpts**

```
vert = extrpts(A, rel, b)
```

```
vert =
         0         0    1.1111
         0    1.0000    2.1111
```

Recall that the extreme points are stored in columns of the matrix **vert**.

Extreme directions, if any, of the polyhedral set **X** can be computed using function **extrdir**

```
function d = extrdir(A, rel, b)

% Extreme directions d of the polyhedral set

%            X = {x: Ax <= b, or Ax >= b, x >= 0}.

% Matrix A must be of the full row rank.

[m, n] = size(A);
n1 = n;
for i=1:m
   if(rel(i) == '>')
      A = [A -vr(m,i)];
   else
      A = [A vr(m,i)];
   end
end
[m, n] = size(A);
A = [A;ones(1,n)];
b = [zeros(m,1);1];
d = feassol(A,b);
if ~isempty(d)
   d1 = d(1:n1,:);
   d = delcols(d1);
   s = sum(d);
   for i=1:n1
      d(:,i) = d(:,i)/s(i);
   end
else
   d = [];
end
```

Function **extrdir** returns the empty set operator **[ ]** for bounded polyhedral sets.

We will test this function using the polyhedral set that is defined in the last example of this section. We have

```
d = extrdir(A, rel, b)

d =
    1.0000    0.9091
         0    0.0909
```

Again, the directions in question are stored in columns of the output matrix **d**.

6.6 Solving the LP problem geometrically

A solution to the LP problem with two legitimate variables x_1 and x_2 can be found geometrically in three steps. First, the feasible region described by the constraint system $Ax \leqslant b$ or $Ax \geqslant b$ with $x \geqslant 0$ is drawn. Next, a direction of the level line $z = c_1x_1 + c_2x_2$ is determined. Finally moving the level line one can find easily vertex (extreme point) of the feasible region where the minimum or maximum value of z is attained or conclude that the objective function is unbounded. For more details see, e.g., [3], Chapter 2 and [1], Chapter 1.

Function **drawfr** implements two initial steps of this method

```
function  drawfr(c, A, rel, b)

% Graphs of the feasible region and the line level
% of the LP problem with two legitimate variables
%
%                    min (max)z = c*x
%                Subject to    Ax <= b (or Ax >= b),
%                               x >= 0

clc
[m, n] = size(A);
if n ~= 2
   str = 'Number of the legitimate variables must be equal to 2';
   msgbox(str,'Error Window','error')
   return
end
vert = extrpts(A,b,rel);
if isempty(vert)
   disp(sprintf('\n   Empty feasible region'))
   return
end
vert = vert(1:2,:);
vert = delcols(vert);
d = extrdir(A,b,rel);
if ~isempty(d)
   msgbox('Unbounded feasible region','Warning Window','warn')
   disp(sprintf('\n   Extreme direction(s) of the constraint set'))
   d
   disp(sprintf('\n   Extreme points of the constraint set'))
   vert
   return
end
t1 = vert(1,:);
t2 = vert(2,:);
z = convhull(t1,t2);
hold on
patch(t1(z),t2(z),'r')
h = .25;
mit1 = min(t1)-h;
mat1 = max(t1)+h;
mit2 = min(t2)-h;
```

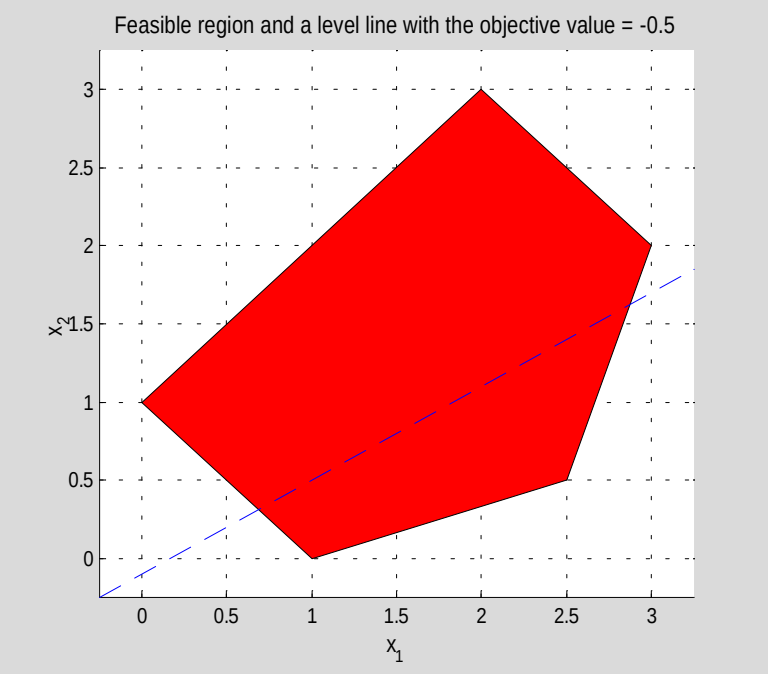
```
mat2 = max(t2)+h;
if c(1) ~= 0 & c(2) ~= 0
   sl = -c(1)/c(2);
   if sl > 0
      z = c(:)'*[mit1;mit2];
      a1 = [mit1 mat1];
      b1 = [mit2 (z-c(1)*mat1)/c(2)];
   else
      z = c(:)'*[mat1;mit2];
      a1 = [mit1 mat1];
      b1 = [(z-c(1)*mit1)/c(2) mit2];
   end
elseif c(1) == 0 & c(2) ~= 0
   z = 0;
   a1 = [mit1 mat1];
   b1 = [0,0];
else
   z = 0;
   a1 = [0 0];
   b1 = [mit2 mat2];
end
h = plot(a1,b1);
set(h,'linestyle','--')
set(h,'linewidth',2)
str = 'Feasible region and a level line with the objective value = ';
title([str,num2str(z)])
axis([mit1 mat1 mit2 mat2])
h = get(gca,'Title');
set(h,'FontSize',11)
xlabel('x_1')
h = get(gca,'xlabel');
set(h,'FontSize',11)
ylabel('x_2')
h = get(gca,'ylabel');
set(h,'FontSize',11)
grid
hold off
```

To test this function consider the LP problem with five constraints

```
c = [-3 5];

A = [-1 1;1 1;1 1;3 -1;1 -3];

rel = '<><<<';

b = [1;1;5;7;1];
```

Graphs of the feasible region and the level line are shown below

```
drawfr(c, A, rel, b)
```



Let us note that for the minimization problem the optimal solution occurs at

$$\mathbf{x} = \begin{bmatrix} 2.5 \\ 0.5 \end{bmatrix}$$

while the maximum of the objective function is attained at

$$\mathbf{x} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

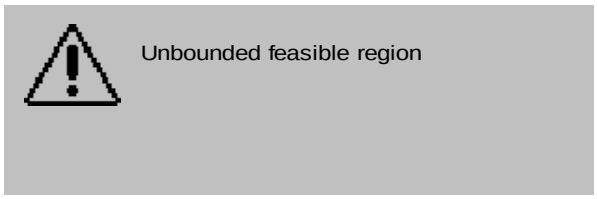
Next LP problem is defined as follows

```
c = [1 1];
A = [-1 1;-0.1 1];
rel = '<<';
b = [1;2];
```

Invoking function **drawfr** we obtain

```
drawfr(c, A, rel, b)
```

```
   Extreme direction(s) of the constraint set
d =
    1.0000    0.9091
         0    0.0909

   Extreme points of the constraint set
vert =
         0         0    1.1111
         0    1.0000    2.1111
```



Graph of the feasible region is not displayed. Extreme directions and the extreme points are shown above. Also, a warrning message is generated.

6.7 The Two-Phase Method

The Two-Phase Method is one of the basic computational tools used in linear programming. An implementation of this method presented in this section takes advantage of all features of this method. The LP problems that can be solved are either the minimization or maximization problems. Optimal solution is sought over a polyhedral set that is described by the inequality and/or equality constraints together with the nonnegativity constraints imposed on the legitimate variables.

Function **simplex2p** displays messages about the LP problem to be solved. These include:

- information about uniqueness of the optimal solution
- information about unboundedness of the objective function, if any
- information about empty feasible region

In addition to these features some built-in mechanisms allow to solve the LP problems whose constraint systems are *redundant*. Also, *Bland's Rule* to prevent *cycling* is used. For details, see [1], p. 169 and p. 174.

During the execution of function **simplex2p** a user has an option to monitor progress of computations by clicking on the **Yes** button in the message windows.

```
function  simplex2p(type, c, A, rel, b)

% The Two-Phase Method for solving the LP problem

%                     min(or max) z = c*x
%                     Subject to   Ax rel b
%                                   x >= 0

% The input parameter type holds information about type of the LP
% problem to be solved. For the minimization problem type = 'min' and
% for the maximization problem type = 'max'.
% The input parameter rel is a string holding the relation signs.
% For instance, if rel = '<=>', then the constraint system consists
% of one inequality <=, one equation, and one inequality >=.

clc
if (type == 'min')
   mm = 0;
else
   mm = 1;
   c = -c;
end
b = b(:);
c = c(:)';
[m, n] = size(A);
n1 = n;
les = 0;
neq = 0;
red = 0;
if length(c) < n
   c = [c zeros(1,n-length(c))];
end
for i=1:m
   if(rel(i) == '<')
      A = [A vr(m,i)];
      les = les + 1;
   elseif(rel(i) == '>')
      A = [A -vr(m,i)];
   else
      neq = neq + 1;
   end
end
ncol = length(A);
if les == m
   c = [c zeros(1,ncol-length(c))];
   A = [A;c];
   A = [A [b;0]];
   [subs, A, z, p1] = loop(A, n1+1:ncol, mm, 1, 1);
   disp('                    End of Phase 1')
   disp('          *********************************')
else
   A = [A eye(m) b];
   if m > 1
      w = -sum(A(1:m,1:ncol));
   else
      w = -A(1,1:ncol);
   end
```

```
   c = [c zeros(1,length(A)-length(c))];
   A = [A;c];
   A = [A;[w zeros(1,m) -sum(b)]];
   subs = ncol+1:ncol+m;
   av = subs;
   [subs, A, z, p1] = loop(A, subs, mm, 2, 1);
   if p1 == 'y'
         disp('                    End of Phase 1')
         disp('          *********************************')
      end
   nc = ncol + m + 1;
   x = zeros(nc-1,1);
   x(subs) = A(1:m,nc);
   xa = x(av);
   com = intersect(subs,av);
   if (any(xa) ~= 0)
      disp(sprintf('\n\n                      Empty feasible region\n'))
      return
   else
      if ~isempty(com)
         red = 1;
      end
   end
   A = A(1:m+1,1:nc);
   A =[A(1:m+1,1:ncol) A(1:m+1,nc)];
   [subs, A, z, p1] = loop(A, subs, mm, 1, 2);
   if p1 == 'y'
         disp('                    End of Phase 2')
         disp('          *********************************')
      end
end
if (z == inf | z == -inf)
   return
end
[m, n] = size(A);
x = zeros(n,1);
x(subs) = A(1:m-1,n);
x = x(1:n1);
if mm == 0
   z = -A(m,n);
else
   z = A(m,n);
end
disp(sprintf('\n\n           Problem has a finite optimal solution\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:n1
   disp(sprintf(' x(%d)= %f ',i,x(i)))
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',z))
t = find(A(m,1:n-1) == 0);
if length(t) > m-1
   str = 'Problem has infinitely many solutions';
   msgbox(str,'Warning Window','warn')
end
if red == 1
   disp(sprintf('\n Constraint system is redundant\n\n'))
```

```
end


function [subs, A, z, p1]= loop(A, subs, mm, k, ph)

% Main loop of the simplex primal algorithm.
% Bland's rule to prevente cycling is used.

tbn = 0;
str1 = 'Would you like to monitor the progress of Phase 1?';
str2 = 'Would you like to monitor the progress of Phase 2?';
if ph == 1
   str = str1;
else
   str = str2;
end
question_ans = questdlg(str,'Make a choice Window','Yes','No','No');
if strcmp(question_ans,'Yes')
   p1 = 'y';
end
if p1 == 'y' & ph == 1
   disp(sprintf('\n\n                    Initial tableau'))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end
if p1 == 'y' & ph == 2
   tbn = 1;
   disp(sprintf('\n\n                    Tableau %g',tbn))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end
[m, n] = size(A);
[mi, col] = Br(A(m,1:n-1));
while ~isempty(mi) & mi < 0 & abs(mi) > eps
   t = A(1:m-k,col);
   if all(t <= 0)
      if mm == 0
         z = -inf;
      else
         z = inf;
      end
      disp(sprintf('\n        Unbounded optimal solution with z=
%s\n',z))
      return
   end
   [row, small] = MRT(A(1:m-k,n),A(1:m-k,col));
   if ~isempty(row)
      if abs(small) <= 100*eps & k == 1
         [s,col] = Br(A(m,1:n-1));
      end
      if p1 == 'y'
         disp(sprintf('          pivot row-> %g   pivot column->
%g',...
            row,col))
      end
```

```
      A(row,:)= A(row,:)/A(row,col);
      subs(row) = col;
      for i = 1:m
         if i ~= row
            A(i,:)= A(i,:)-A(i,col)*A(row,:);
         end
      end
      [mi, col] = Br(A(m,1:n-1));
   end
   tbn = tbn + 1;
   if p1 == 'y'
      disp(sprintf('\n\n                    Tableau %g',tbn))
      A
      disp(sprintf(' Press any key to continue ...\n\n'))
      pause
   end
end
z = A(m,n);
```

MATLAB subfunction **loop** is included in the m-file **simplex2p**. It implements the main loop of the simplex algorithm. Organization of the tableaux generated by function **simplex2p** is the same as the one presented in [3].

We will now test this function on five LP problems.

Let

```
type = 'min';

c = [-3 4];

A = [1 1;2 3];

rel = '<>';

b = [4; 18];
```

One can easily check, either by drawing the graph of the constraint set or running function **drawfr** on these data, that this problem has an empty feasible region.

```
simplex2p(type, c, A, rel, b)
```

```
                Initial tableau

A =

     1     1     1     0     1     0     4
     2     3     0    -1     0     1    18
    -3     4     0     0     0     0     0
    -3    -4    -1     1     0     0   -22

 Press any key to continue ...
```

```
          pivot row-> 1   pivot column-> 1

                    Tableau 1

A =

     1     1     1     0     1     0     4
     0     1    -2    -1    -2     1    10
     0     7     3     0     3     0    12
     0    -1     2     1     3     0   -10

 Press any key to continue ...

          pivot row-> 1   pivot column-> 2

                    Tableau 2

A =

     1     1     1     0     1     0     4
    -1     0    -3    -1    -3     1     6
    -7     0    -4     0    -4     0   -16
     1     0     3     1     4     0    -6

 Press any key to continue ...

                End of Phase 1
          *********************************

                    Empty feasible region
```

Next LP problem has an unbounded objective function. Let

```
type = 'max';

c = [3 2 1];

A = [2 –3 2;-1 1 1];

rel = '<<';

b = [3;55];
```

Running function **simplex2p** we obtain

```
simplex2p(type, c, A, rel, b)
```

```
                    Initial tableau

A =

     2    -3     2     1     0     3
    -1     1     1     0     1    55
    -3    -2    -1     0     0     0

 Press any key to continue ...


          pivot row-> 1   pivot column-> 1


                    Tableau 1

A =

    1.0000   -1.5000    1.0000    0.5000         0    1.5000
         0   -0.5000    2.0000    0.5000    1.0000   56.5000
         0   -6.5000    2.0000    1.5000         0    4.5000

 Press any key to continue ...


        Unbounded optimal solution with z= Inf

                  End of Phase 1
          *********************************
```

In this example we will deal with the LP problem that is described by a system of two equations with seven variables

```
type = 'min';

c = [3 4 6 7 1 0 0];

A = [2 –1 1 6 –5 –1 0;1 1 2 1 2 0 –1];

rel = '==';

b = [6;3];


simplex2p(type, c, A, rel, b)
```

```
                    Initial tableau

A =

     2    -1     1     6    -5    -1     0     1     0     6
     1     1     2     1     2     0    -1     0     1     3
     3     4     6     7     1     0     0     0     0     0
    -3     0    -3    -7     3     1     1     0     0    -9

 Press any key to continue ...


          pivot row-> 1   pivot column-> 1


                   Tableau 1

A =

  Columns 1 through 7

    1.0000   -0.5000    0.5000    3.0000   -2.5000   -0.5000         0
         0    1.5000    1.5000   -2.0000    4.5000    0.5000   -1.0000
         0    5.5000    4.5000   -2.0000    8.5000    1.5000         0
         0   -1.5000   -1.5000    2.0000   -4.5000   -0.5000    1.0000

  Columns 8 through 10

    0.5000         0    3.0000
   -0.5000    1.0000         0
   -1.5000         0   -9.0000
    1.5000         0         0

 Press any key to continue ...


          pivot row-> 2   pivot column-> 2
```

```
                    Tableau 2

A =

  Columns 1 through 7

    1.0000         0    1.0000    2.3333   -1.0000   -0.3333   -0.3333
         0    1.0000    1.0000   -1.3333    3.0000    0.3333   -0.6667
         0         0   -1.0000    5.3333   -8.0000   -0.3333    3.6667
         0         0         0         0         0         0         0

  Columns 8 through 10

    0.3333    0.3333    3.0000
   -0.3333    0.6667         0
    0.3333   -3.6667   -9.0000
    1.0000    1.0000         0

 Press any key to continue ...


                End of Phase 1
          ********************************


                    Tableau 1

A =

  Columns 1 through 7

    1.0000         0    1.0000    2.3333   -1.0000   -0.3333   -0.3333
         0    1.0000    1.0000   -1.3333    3.0000    0.3333   -0.6667
         0         0   -1.0000    5.3333   -8.0000   -0.3333    3.6667

  Column 8

    3.0000
         0
   -9.0000

 Press any key to continue ...


          pivot row-> 2   pivot column-> 3
```

```
                    Tableau 2

A =

  Columns 1 through 7

    1.0000   -1.0000         0    3.6667   -4.0000   -0.6667    0.3333
         0    1.0000    1.0000   -1.3333    3.0000    0.3333   -0.6667
         0    1.0000         0    4.0000   -5.0000    0.0000    3.0000

  Column 8

    3.0000
         0
   -9.0000

 Press any key to continue ...


          pivot row-> 2   pivot column-> 5


                    Tableau 3

A =

  Columns 1 through 7

    1.0000    0.3333    1.3333    1.8889         0   -0.2222   -0.5556
         0    0.3333    0.3333   -0.4444    1.0000    0.1111   -0.2222
         0    2.6667    1.6667    1.7778         0    0.5556    1.8889

  Column 8

    3.0000
         0
   -9.0000

 Press any key to continue ...


                    End of Phase 2
          *********************************


           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 3.000000
 x(2)= 0.000000
 x(3)= 0.000000
 x(4)= 0.000000
 x(5)= 0.000000
 x(6)= 0.000000
 x(7)= 0.000000
```

```
 Objective value at the optimal point:

 z= 9.000000
```

The constraint system of the following LP problem

```
type = 'min';

c = [-1 2 –3];

A = [1 1 1;-1 1 2;0 2 3;0 0 1];

rel = '===<';

b = [6 4 10 2];
```

is redundant. Function **simplex2p** generates the following tableaux

```
simplex2p(type, c, A, rel, b)

                    Initial tableau

A =

     1     1     1     0     1     0     0     0     6
    -1     1     2     0     0     1     0     0     4
     0     2     3     0     0     0     1     0    10
     0     0     1     1     0     0     0     1     2
    -1     2    -3     0     0     0     0     0     0
     0    -4    -7    -1     0     0     0     0   -22

 Press any key to continue ...

          pivot row-> 2   pivot column-> 2

                    Tableau 1

A =

     2     0    -1     0     1    -1     0     0     2
    -1     1     2     0     0     1     0     0     4
     2     0    -1     0     0    -2     1     0     2
     0     0     1     1     0     0     0     1     2
     1     0    -7     0     0    -2     0     0    -8
    -4     0     1    -1     0     4     0     0    -6

 Press any key to continue ...

          pivot row-> 1   pivot column-> 1
```

```
                    Tableau 2

A =

  Columns 1 through 7

    1.0000         0   -0.5000         0    0.5000   -0.5000         0
         0    1.0000    1.5000         0    0.5000    0.5000         0
         0         0         0         0   -1.0000   -1.0000    1.0000
         0         0    1.0000    1.0000         0         0         0
         0         0   -6.5000         0   -0.5000   -1.5000         0
         0         0   -1.0000   -1.0000    2.0000    2.0000         0

  Columns 8 through 9

         0    1.0000
         0    5.0000
         0         0
    1.0000    2.0000
         0   -9.0000
         0   -2.0000

 Press any key to continue ...


          pivot row-> 4   pivot column-> 3


                    Tableau 3

A =

  Columns 1 through 7

    1.0000         0         0    0.5000    0.5000   -0.5000         0
         0    1.0000         0   -1.5000    0.5000    0.5000         0
         0         0         0         0   -1.0000   -1.0000    1.0000
         0         0    1.0000    1.0000         0         0         0
         0         0         0    6.5000   -0.5000   -1.5000         0
         0         0         0         0    2.0000    2.0000         0

  Columns 8 through 9

    0.5000    2.0000
   -1.5000    2.0000
         0         0
    1.0000    2.0000
    6.5000    4.0000
    1.0000         0

 Press any key to continue ...


                End of Phase 1
          ********************************
```

```
                    Tableau 1

A =

    1.0000         0         0    0.5000    2.0000
         0    1.0000         0   -1.5000    2.0000
         0         0         0         0         0
         0         0    1.0000    1.0000    2.0000
         0         0         0    6.5000    4.0000

 Press any key to continue ...


               End of Phase 2
          ********************************


           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 2.000000
 x(2)= 2.000000
 x(3)= 2.000000
 Objective value at the optimal point:

 z= -4.000000

 Constraint system is redundant
```

If the LP problem is *degenerated*, then there is a chance that the method under discussion would never terminate. In order to avoid cycling the Bland's rule is implemented in the body of the function l**oop**. The following LP problem

```
type = 'min';

c = [-3/4 150 –1/50 6];

A = [1/4 –60 –1/25 9; 1/2 -90 –1/50 3; 0 0 1 0];

rel = '<<<';

b = [0 0 1];
```

is discussed in [2], pp. 136-138. Function **simplex2p** generates the following tableaux

```
simplex2p(type, c, A, rel, b)
```

```
                    Initial tableau

A =

  Columns 1 through 7

    0.2500  -60.0000   -0.0400    9.0000    1.0000         0         0
    0.5000  -90.0000   -0.0200    3.0000         0    1.0000         0
         0         0    1.0000         0         0         0    1.0000
   -0.7500  150.0000   -0.0200    6.0000         0         0         0

  Column 8

         0
         0
    1.0000
         0

 Press any key to continue ...


          pivot row-> 1   pivot column-> 1


                   Tableau 1

A =

  Columns 1 through 7

    1.0000 -240.0000   -0.1600   36.0000    4.0000         0         0
         0   30.0000    0.0600  -15.0000   -2.0000    1.0000         0
         0         0    1.0000         0         0         0    1.0000
         0  -30.0000   -0.1400   33.0000    3.0000         0         0

  Column 8

         0
         0
    1.0000
         0

 Press any key to continue ...


          pivot row-> 2   pivot column-> 2
```

```
                    Tableau 2

A =

  Columns 1 through 7

    1.0000         0    0.3200  -84.0000  -12.0000    8.0000         0
         0    1.0000    0.0020   -0.5000   -0.0667    0.0333         0
         0         0    1.0000         0         0         0    1.0000
         0         0   -0.0800   18.0000    1.0000    1.0000         0

  Column 8

         0
         0
    1.0000
         0

 Press any key to continue ...


          pivot row-> 1   pivot column-> 3


                    Tableau 3

A =

  Columns 1 through 7

    3.1250         0    1.0000 -262.5000  -37.5000   25.0000         0
   -0.0063    1.0000         0    0.0250    0.0083   -0.0167         0
   -3.1250         0         0  262.5000   37.5000  -25.0000    1.0000
    0.2500         0         0   -3.0000   -2.0000    3.0000         0

  Column 8

         0
         0
    1.0000
         0

 Press any key to continue ...


          pivot row-> 2   pivot column-> 4
```

```
                    Tableau 4

A =

  1.0e+004 *

  Columns 1 through 7

   -0.0062    1.0500    0.0001         0    0.0050   -0.0150         0
   -0.0000    0.0040         0    0.0001    0.0000   -0.0001         0
    0.0062   -1.0500         0         0   -0.0050    0.0150    0.0001
   -0.0000    0.0120         0         0   -0.0001    0.0001         0

  Column 8

         0
         0
    0.0001
         0

 Press any key to continue ...


          pivot row-> 3   pivot column-> 1


                    Tableau 5

A =

  Columns 1 through 7

         0         0    1.0000         0         0         0    1.0000
         0   -2.0000         0    1.0000    0.1333   -0.0667    0.0040
    1.0000 -168.0000         0         0   -0.8000    2.4000    0.0160
         0   36.0000         0         0   -1.4000    2.2000    0.0080

  Column 8

    1.0000
    0.0040
    0.0160
    0.0080

 Press any key to continue ...


          pivot row-> 2   pivot column-> 5
```

```
                    Tableau 6

A =

  Columns 1 through 7

         0         0    1.0000         0         0         0    1.0000
         0  -15.0000         0    7.5000    1.0000   -0.5000    0.0300
    1.0000 -180.0000         0    6.0000         0    2.0000    0.0400
         0   15.0000         0   10.5000         0    1.5000    0.0500

  Column 8

    1.0000
    0.0300
    0.0400
    0.0500

 Press any key to continue ...


                End of Phase 1
          ********************************


           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 0.040000
 x(2)= 0.000000
 x(3)= 1.000000
 x(4)= 0.000000

 Objective value at the optimal point:

 z= -0.050000
```

The last entry in column eight, in tableaux 1 through 4, is equal to zero. Recall that this is the current value of the objective function. Simplex algorithm tries to improve a current basic feasible solution and after several attempts it succeeds. Two basic variables in tableaux 1 through 4 are both equal to zero. Based on this observation one can conclude that the problem in question is degenerated. Without a built-in mechanism that prevents cycling this algorithm would never terminate. Another example of the degenerated LP problem is discussed in [3], Appendix C, pp. 375-376.

6.8 The Dual Simplex Algorithm

Another method that is of great importance in linear programming is called the *Dual Simplex Algorithm*. The optimization problem that can be solved with the aid of this method is formulated as follows

$$\begin{aligned} \textbf{min(max)} \quad & z = c^T x \\ \textbf{Subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Components of the vector **b** are not required to satisfy the nonnegativity constraints. The Dual Simplex Algorithm has numerous applications to other problems of linear programming. It is used, for instance, in some implementations of the Gomory's cutting plane algorithm for solving the integer programming problems.

Function **dsimplex** computes an optimal solution to the optimization problem that is defined above. One of the nice features of this method is its ability to detect the case of the empty feasible region. Function under discussion takes four input parameters **types**, **c**, **A** and **b**. Their meaning is described in Section 6.7 of this tutorial.

```
function varargout = dsimplex(type, c, A, b)

% The Dual Simplex Algorithm for solving the LP problem

%                      min (max) z = c*x
%                    Subject to  Ax >= b
%                                 x >= 0
%

if type == 'min'
   mm = 0;
else
   mm = 1;
   c = -c;
end
str = 'Would you like to monitor the progress of computations?';
A = -A;
b = -b(:);
c = c(:)';
[m, n] = size(A);
A = [A eye(m) b];
A = [A;[c zeros(1,m+1)]];
question_ans = questdlg(str,'Make a choice Window','Yes','No','No');
if strcmp(question_ans,'Yes')
   p1 = 'y';
else
   p1 = 'n';
end
if p1 == 'y'
   disp(sprintf('\n\n                    Initial tableau'))
   A
```

```
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end
subs = n+1:n+m;
[bmin, row] = Br(b);
tbn = 0;
while ~isempty(bmin) & bmin < 0 & abs(bmin) > eps
   if A(row,1:m+n) >= 0
      disp(sprintf('\n\n               Empty feasible region\n'))
      varargout(1)={subs(:)};
      varargout(2)={A};
      varargout(3) = {zeros(n,1)};
      varargout(4) = {0};
      return
   end
   col = MRTD(A(m+1,1:m+n),A(row,1:m+n));
   if p1 == 'y'
      disp(sprintf('          pivot row-> %g   pivot column-> %g',...
            row,col))
   end
   subs(row) = col;
   A(row,:)= A(row,:)/A(row,col);
      for i = 1:m+1
      if i ~= row
         A(i,:)= A(i,:)-A(i,col)*A(row,:);
      end
   end
   tbn = tbn + 1;
   if p1 == 'y'
      disp(sprintf('\n\n                    Tableau %g',tbn))
      A
      disp(sprintf(' Press any key to continue ...\n\n'))
      pause
   end
   [bmin, row] = Br(A(1:m,m+n+1));
end
x = zeros(m+n,1);
x(subs) = A(1:m,m+n+1);
x = x(1:n);
if mm == 0
   z = -A(m+1,m+n+1);
else
   z = A(m+1,m+n+1);
end
disp(sprintf('\n\n           Problem has a finite optimal
solution\n\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:n
   disp(sprintf(' x(%d)= %f ',i,x(i)))
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',z))
disp(sprintf('\n Indices of basic variables in the final tableau:'))
varargout(1)={subs(:)};
varargout(2)={A};
varargout(3) = {x};
varargout(4) = {z};
```

Function **dsimplex** allows a variable number of the output parameters. Indices of the basic variables in the final tableau are always displayed. The final tableau also can be saved in a variable. For instance, execution of the following command **[subs, B] = dsimplex(type, c, A, b)** will return, among other things, indices of basic variables stored in the variable **subs** and the final tableau stored in the variable **B**.

Let

```
type = 'min';

c = [-2 3 4];

A = [-1 –1 –1;2 2 2];

b = [-1;4];
```

One can easily check that the constraint system defines an empty set. Running function **dsimplex**, we obtain

```
subs = dsimplex(type, c, A, b)


          Initial tableau

A =

     1     1     1     1     0     1
    -2    -2    -2     0     1    -4
    -2     3     4     0     0     0

 Press any key to continue ...


     pivot row-> 2   pivot column-> 1


                   Tableau 1

A =

         0         0         0    1.0000    0.5000   -1.0000
    1.0000    1.0000    1.0000         0   -0.5000    2.0000
         0    5.0000    6.0000         0   -1.0000    4.0000

 Press any key to continue ...


               Empty feasible region
```

```
subs =

     4
     1
```

In this example we will use function **dsimplex** to find an optimal solution to the problem with four variables and three constraints

```
type = 'max';

c = [1 2 3 –1];

A = [2 1 5 0;1 2 3 0;1 1 1 1];

b = [20;25;10];


subs = dsimplex(type, c, A, b)
```

```
                    Initial tableau

A =

    -2    -1    -5     0     1     0     0   -20
    -1    -2    -3     0     0     1     0   -25
    -1    -1    -1    -1     0     0     1   -10
    -1    -2    -3     1     0     0     0     0
 Press any key to continue ...


          pivot row-> 1   pivot column-> 2


                    Tableau 1

A =

     2     1     5     0    -1     0     0    20
     3     0     7     0    -2     1     0    15
     1     0     4    -1    -1     0     1    10
     3     0     7     1    -2     0     0    40

 Press any key to continue ...
```

```
           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 0.000000
 x(2)= 20.000000
 x(3)= 0.000000
 x(4)= 0.000000

 Objective value at the optimal point:

 z= 40.000000

 Indices of basic variables in the final tableau:

subs =

     2
     6
     7
```

6.9 Addition of a constraint

Topic discussed in this section is of interest in the sensitivity analysis. Suppose that the LP problem has been solved using one of the methods discussed earlier in this tutorial. We assume that the final tableau **A** and the vector **subs** - vector of indices of basic variables are given. One wants to find an optimal solution to the original problem with an extra constraint $\mathbf{a}^T\mathbf{x} \leqslant \mathbf{d}$ or $\mathbf{a}^T\mathbf{x} \geqslant \mathbf{d}$ added.

Function **addconstr(type, A, subs, a, rel, d)** implements a method that is described in [3], pp. 171-174.

```
function addconstr(type, A, subs, a, rel, d)

% Adding a constraint to the final tableau A.
% The input parameter subs holds indices of basic
% variables associated with tableau A. Parameters
% lhs, rel, and rhs hold information about a costraint
% to be added:
% a - coefficients of legitimate variables
% rel - string describing the inequality sign;
% for instance, rel = '<' stands for <=.
% d - constant term of the constraint to be added.
% Parameter type is a string that describes type of
% the optimization problem. For the minimization problems
% type = 'min' and for the maximization problems type = 'max'.


clc
str = 'Would you like to monitor the progress of computations?';
question_ans = questdlg(str,'Make a choice Window','Yes','No','No');
if strcmp(question_ans,'Yes')
```

```
   p1 = 'y';
else
   p1 = 'n';
end
[m, n] = size(A);
a = a(:)';
lc = length(a);
if lc < n-1
   a = [a zeros(1,n-lc-1)];
end
if type == 'min'
   ty = -1;
else
   ty = 1;
end
x = zeros(n-1,1);
x(subs) = A(1:m-1,n);
dp = a*x;
if (dp <= d & rel == '<') | (dp >= d & rel == '>')
   disp(sprintf('\n\n            Problem has a finite optimal
solution\n'))
   disp(sprintf('\n Values of the legitimate variables:\n'))
   for i=1:n-1
      disp(sprintf(' x(%d)= %f ',i,x(i)))
   end
   disp(sprintf('\n Objective value at the optimal point:\n'))
   disp(sprintf(' z= %f',ty*A(m,n)))
   return
end
B = [A(:,1:n-1) zeros(m,1) A(:,n)];
if rel == '<'
   a = [a 1 d];
else
   a = [a -1 d];
end
for i=1:m-1
   a = a - a(subs(i))*B(i,:);
end
if a(end) > 0
   a = -a;
end
A = [B(1:m-1,:);a;B(m,:)];
if p1 == 'y'
   disp(sprintf('\n\n                    Initial tableau'))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end
[bmin, row] = Br(A(1:m,end));
tbn = 0;
while ~isempty(bmin) & bmin < 0 & abs(bmin) > eps
   if A(row,1:n) >= 0
      disp(sprintf('\n\n               Empty feasible region\n'))
      return
   end
   col = MRTD(A(m+1,1:n),A(row,1:n));
   if p1 == 'y'
```

```
         disp(sprintf('          pivot row-> %g   pivot column->
%g',...
            row,col))
   end
   subs(row) = col;
   A(row,:)= A(row,:)/A(row,col);
      for i = 1:m+1
      if i ~= row
         A(i,:)= A(i,:)-A(i,col)*A(row,:);
      end
   end
   tbn = tbn + 1;
   if p1 == 'y'
      disp(sprintf('\n\n                    Tableau %g',tbn))
      A
      disp(sprintf(' Press any key to continue ...\n\n'))
      pause
   end
  [bmin, row] = Br(A(1:m,end));
end
x = zeros(n+1,1);
x(subs) = A(1:m,end);
x = x(1:n);
disp(sprintf('\n\n           Problem has a finite optimal
solution\n\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:n
   disp(sprintf(' x(%d)= %f ',i,x(i)))
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',ty*A(m+1,n+1)))
```

The following examples are discussed in [3], pp. 171-173. Let

```
type = 'max';

A = [-2 0 5 1 2 –1 6; 11 1 –18 0 –7 4 4; 3 0 2 0 2 1 106];

subs = [4; 2];

a = [4 1 0 4];

rel = '>';

d = 29;


addconstr(type, A, subs, a, rel, d)
```

```
                    Initial tableau

A =

    -2     0     5     1     2    -1     0     6
    11     1   -18     0    -7     4     0     4
    -1     0     2     0     1     0     1    -1
     3     0     2     0     2     1     0   106

 Press any key to continue ...

          pivot row-> 3   pivot column-> 1

                 Tableau 1

A =

     0     0     1     1     0    -1    -2     8
     0     1     4     0     4     4    11    -7
     1     0    -2     0    -1     0    -1     1
     0     0     8     0     5     1     3   103

 Press any key to continue ...

               Empty feasible region
```

Changing the additional constraint to $3x_1 + x_2 + 3x_4 \leqslant 20$, we obtain

```
a = [3 1 0 3];

rel = '<';

d = 20;

addconstr(type, A, subs, a, rel, d)


                    Initial tableau
A =
    -2     0     5     1     2    -1     0     6
    11     1   -18     0    -7     4     0     4
    -2     0     3     0     1    -1     1    -2
     3     0     2     0     2     1     0   106
 Press any key to continue ...

          pivot row-> 3   pivot column-> 6
```

```
                    Tableau 1
A =
     0     0     2     1     1     0    -1     8
     3     1    -6     0    -3     0     4    -4
     2     0    -3     0    -1     1    -1     2
     1     0     5     0     3     0     1   104
 Press any key to continue ...


          pivot row-> 2   pivot column-> 3

                    Tableau 2
A =
  Columns 1 through 7
    1.0000    0.3333         0    1.0000         0         0    0.3333
   -0.5000   -0.1667    1.0000         0    0.5000         0   -0.6667
    0.5000   -0.5000         0         0    0.5000    1.0000   -3.0000
    3.5000    0.8333         0         0    0.5000         0    4.3333
  Column 8
    6.6667
    0.6667
    4.0000
  100.6667
 Press any key to continue ...


           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 0.000000
 x(2)= 0.000000
 x(3)= 0.666667
 x(4)= 6.666667
 x(5)= 0.000000
 x(6)= 4.000000
 x(7)= 0.000000

 Objective value at the optimal point:

 z= 100.666667
```

6.10 Gomory's cutting plane algorithm

Some problems that arise in economy can be formulated as the linear programming problems with decision variables being restricted to integral values

min(max) z = c*x
Subject to Ax ⩽ b
x ⩾ 0 and integral

where the vector of the right-hand sides **b** is nonnegative. Among numerous algorithms designed for solving this problem the one, proposed by R.E. Gomory (see [3], pp. 194-203) is called the *cutting plane algorithm*. Its implementation, named here **cpa**, is included in this tutorial

```
function cpa(type, c, A, b)

% Gomory's cutting plane algorithm for solving
% the integer programming problem

%                      min(max) z = c*x
%                   Subject to: Ax <= b
%                                x >= 0 and integral


str = 'Would you like to monitor the progress of computations?';
question_ans = questdlg(str,'Make a choice Window','Yes','No','No');
if strcmp(question_ans,'Yes')
   p1 = 'y';
else
   p1 = 'n';
end
if type == 'min'
   tp = -1;
else
   tp = 1;
end
[m,n] = size(A);
nlv = n;
b = b(:);
c = c(:)';
if p1 == 'y'
   [A,subs] = simplex(type,c,A,b,p1);
else
   [A,subs] = simplex(type,c,A,b);
end
[m,n] = size(A);
d = A(1:m-1,end);
pc = fractp(d);
tbn = 0;
if p1 == 'y'
   disp(sprintf('
_____________________________________________________'))
   disp(sprintf('\n              Tableaux of the Dual Simplex Method'))
   disp(sprintf('
_____________________________________________________'))
end
while norm(pc,'inf') > eps
   [el,i] = max(pc);
   nr = A(i,1:n-1);
   nr = [-fractp(nr) 1 -el];
   B = [A(1:m-1,1:n-1) zeros(m-1,1) A(1:m-1,end)];
   B = [B;nr;[A(m,1:n-1) 0 A(end,end)]];
   A = B;
   [m,n] = size(A);
```

```
   [bmin, row] = min(A(1:m-1,end));
   while bmin < 0 & abs(bmin) > eps
      col = MRTD(A(m,1:n-1),A(row,1:n-1));
      if p1 == 'y'
         disp(sprintf('\n          pivot row-> %g   pivot column->
%g',...
               row,col))
         tbn = tbn + 1;
         disp(sprintf('\n                    Tableau %g',tbn))
         A
         disp(sprintf(' Press any key to continue ...\n'))
         pause
      end
      if isempty(col)
         disp(sprintf('\n Algorithm fails to find an optimal
solution.'))
         return
      end
      A(row,:)= A(row,:)./A(row,col);
      subs(row) = col;
      for i = 1:m
         if i ~= row
            A(i,:)= A(i,:)-A(i,col)*A(row,:);
         end
      end
     [bmin, row] = min(A(1:m-1,end));
   end
   d = A(1:m-1,end);
   pc = fractp(d);
end
if p1 == 'y'
   disp(sprintf('\n                  Final tableau'))
   A
   disp(sprintf(' Press any key to continue ...\n'))
   pause
end
x = zeros(n-1,1);
x(subs) = A(1:m-1,end);
x = x(1:nlv);
disp(sprintf('\n           Problem has a finite optimal solution\n\n'))
disp(sprintf('\n Values of the legitimate variables:\n'))
for i=1:nlv
   disp(sprintf(' x(%d)= %g ',i,x(i)))
end
disp(sprintf('\n Objective value at the optimal point:\n'))
disp(sprintf(' z= %f',tp*A(m,n)))



function y = fractp(x)

% Fractional part y of x, where x is a matrix.

y = zeros(1,length(x));
ind = find(abs(x - round(x)) >= 100*eps);
y(ind) = x(ind) - floor(x(ind));
```

The subfunction **fractp** is called from within the function **cpa**. It computes the fractional parts of real numbers that are stored in a matrix. Function **cpa** makes a call to another function called **simplex**. The latter finds an optimal solution to the problem that is formulated in the beginning of this section without the integral restrictions imposed on the legitimate variables.

```
function[A, subs, x, z] = simplex(type, c, A, b, varargin);

% The simplex algorithm for the LP problem

%                    min(max) z = c*x
%                 Subject to: Ax <= b
%                              x >= 0

% Vector b must be nonnegative.
% For the minimization problems the string type = 'min',
% otherwise type = 'max'.
% The fifth input parameter is optional. If it is set to 'y',
% then the initial and final tableaux are displayed to the
% screen.
% Output parameters:
% A - final tableau of the simplex method
% subs - indices of the basic variables in the final tableau
% x - optimal solution
% z - value of the objective function at x.

if any(b < 0)
   error(' Right hand sides of the constraint set must be
nonnegative.')
end
if type == 'min'
   tp = -1;
else
   tp = 1;
   c = -c;
end
[m, n] = size(A);
A = [A eye(m)];
b = b(:);
c = c(:)';
A = [A b];
d = [c zeros(1,m+1)];
A = [A;d];
if nargin == 5
   disp(sprintf('
_____________________________________________________'))
   disp(sprintf('\n              Tableaux of the Simplex Algorithm'))
   disp(sprintf('
_____________________________________________________'))
   disp(sprintf('\n                   Initial tableau\n'))
   A
   disp(sprintf(' Press any key to continue ...\n\n'))
   pause
end
[mi, col] = Br(A(m+1,1:m+n));
subs = n+1:m+n;
```

```
while ~isempty(mi) & mi < 0 & abs(mi) > eps
   t = A(1:m,col);
   if all(t <= 0)
      disp(sprintf('\n       Problem has unbounded objective
function'));
      x = zeros(n,1);
      if tp == -1
         z = -inf;
      else
         z = inf;
      end
      return;
   end
   row = MRT(A(1:m,m+n+1),A(1:m,col));
   if ~isempty(row)
      A(row,:)= A(row,:)/A(row,col);
      subs(row) = col;
      for i = 1:m+1
         if i ~= row
            A(i,:)= A(i,:)-A(i,col)*A(row,:);
         end
      end
   end
   [mi, col] = Br(A(m+1,1:m+n));
end
z = tp*A(m+1,m+n+1);
x = zeros(1,m+n);
x(subs) = A(1:m,m+n+1);
x = x(1:n)';
if nargin == 5
   disp(sprintf('\n\n                    Final tableau'))
   A
   disp(sprintf(' Press any key to continue ...\n'))
   pause
end
```

We will test function **cpa** on two examples from [3], pp. 195-201. Let

```
type = 'min';

c = [1 –3];

A = [1 –1;2 4];

b = [2;15];
```

Using the option to monitor computations, the following tableaux are displayed to the screen

```
cpa(type, c, A, b)
```

```
_____________________________________________________

         Tableaux of the Simplex Algorithm
_____________________________________________________


                Initial tableau

A =

     1    -1     1     0     2
     2     4     0     1    15
     1    -3     0     0     0

 Press any key to continue ...



                Final tableau

A =

    1.5000         0    1.0000    0.2500    5.7500
    0.5000    1.0000         0    0.2500    3.7500
    2.5000         0         0    0.7500   11.2500

 Press any key to continue ...


_____________________________________________________

         Tableaux of the Dual Simplex Method
_____________________________________________________

      pivot row-> 3   pivot column-> 4

                Tableau 1

A =

    1.5000         0    1.0000    0.2500         0    5.7500
    0.5000    1.0000         0    0.2500         0    3.7500
   -0.5000         0         0   -0.2500    1.0000   -0.7500
    2.5000         0         0    0.7500         0   11.2500

 Press any key to continue ...
```

```
                  Final tableau

A =

     1     0     1     0     1     5
     0     1     0     0     1     3
     2     0     0     1    -4     3
     1     0     0     0     3     9

 Press any key to continue ...


           Problem has a finite optimal solution


 Values of the legitimate variables:

 x(1)= 0
 x(2)= 3

 Objective value at the optimal point:

 z= -9.000000
```

Only the initial and final tableaux of the simplex algorithm are displayed. However, the all tableaux generated during the execution of the dual simplex method are included.

Let now

```
type = 'min';

c = [1 -2];

A = [2 1;-4 4];
b = [5;5];

cpa(type, c, A, b)
```

```
      ---------------------------------------------------

              Tableaux of the Simplex Algorithm

      ---------------------------------------------------

                  Initial tableau

A =
     2     1     1     0     5
    -4     4     0     1     5
     1    -2     0     0     0
 Press any key to continue ...
```

```
                Final tableau
A =
    1.0000         0    0.3333   -0.0833    1.2500
         0    1.0000    0.3333    0.1667    2.5000
         0         0    0.3333    0.4167    3.7500
 Press any key to continue ...
```

Tableaux of the Dual Simplex Method

```
          pivot row-> 3   pivot column-> 3

                  Tableau 1
A =
    1.0000         0    0.3333   -0.0833         0    1.2500
         0    1.0000    0.3333    0.1667         0    2.5000
         0         0   -0.3333   -0.1667    1.0000   -0.5000
         0         0    0.3333    0.4167         0    3.7500
 Press any key to continue ...

          pivot row-> 4   pivot column-> 4

                  Tableau 2
A =
    1.0000         0         0   -0.2500    1.0000         0    0.7500
         0    1.0000         0         0    1.0000         0    2.0000
         0         0    1.0000    0.5000   -3.0000         0    1.5000
         0         0         0   -0.7500         0    1.0000   -0.7500
         0         0         0    0.2500    1.0000         0    3.2500
 Press any key to continue ...



                Final tableau
A =
    1.0000         0         0         0    1.0000   -0.3333    1.0000
         0    1.0000         0         0    1.0000         0    2.0000
         0         0    1.0000         0   -3.0000    0.6667    1.0000
         0         0         0    1.0000         0   -1.3333    1.0000
         0         0         0         0    1.0000    0.3333    3.0000
 Press any key to continue ...
```

```
          Problem has a finite optimal solution

 Values of the legitimate variables:

 x(1)= 1
 x(2)= 2

Objective value at the optimal point:

 z= -3.000000
```

Function **cpa** has been tested extensively. It failed, however, to compute the optimal solution to the following integer programming problem (see [3], pp. 208-209)

$$\begin{aligned} \textbf{max} \quad & z = 3x_1 + 13x_2 \\ \textbf{Subject to} \quad & 2x_1 + 9x_2 \leqslant 40 \\ & 11x_1 - 8x_2 \leqslant 82 \\ & x_1, x_2 \geqslant 0 \text{ and integral} \end{aligned}$$

In Problem 17 you are asked to determine the cause of the premature termination of computations.

References

[1] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali, Linear Programming and Network Flows, Second edition, John Wiley & Sons, New York, 1990.

[2] S.G. Nash and A. Sofer, Linear and Nonlinear Programming, The McGraw-Hill Companies, Inc., New York, 1996.

[3] P.R. Thie, An Introduction to Linear Programming and Game Theory, Second edition, John Wiley & Sons, New York, 1988.

Problems

1. Solve the following LP problems geometrically.

(i)
$$\begin{aligned} \textbf{min } z &= -3x_1 + 9x_2 \\ \textbf{Subject to } & -x_1 + x_2 \leqslant 1 \\ & x_1 + x_2 \geqslant 1 \\ & x_1 + x_2 \leqslant 5 \\ & 3x_1 - x_2 \leqslant 7 \\ & x_1 - 3x_2 \leqslant 1 \\ & x_1, x_2 \geqslant 0 \end{aligned}$$

Let the objective function and the constraint set be the same as in Part (i). Find the optimal solution to the maximization problem.

(ii)
$$\begin{aligned} \textbf{max } z &= x_1 + x_2 \\ \textbf{Subject to } & x_1 + x_2 \leqslant 2 \\ & x_1 \leqslant 1 \\ & x_1 + x_2 \geqslant 1 \\ & -x_1 + x_2 \leqslant 1.5 \\ & x_1, x_2 \geqslant 0 \end{aligned}$$

2. Given the constraint set $\mathbf{X = \{(x_1, x_2, x_3)^T \in \mathbb{R}^3 : \ 0 \leqslant x_1 \leqslant x_2 \leqslant x_3\}}$.

(i) Write these constraints in the following form $\mathbf{Ax \leqslant b}$, $\mathbf{x \geqslant 0}$.
(ii) Find the extreme points and the extreme directions, if any, of $\mathbf{X}$.
(iii) Based on the numerical evidence from Part (ii) of this problem, formulate a hypothesis about the extreme points and the extreme directions of the more general constraint set $\mathbf{X = \{(x_1, x_2, \ldots x_n)^T \in \mathbb{R}^n : \ 0 \leqslant x_1 \leqslant x_2 \leqslant \ldots \leqslant x_n\}}$.

3. Give an example of the constraint set with at least three constraints and three variables that have empty feasible region.

4. Consider the system of constraints in the standard form $\mathbf{Ax = b}$, $\mathbf{x \geqslant 0}$ and $\mathbf{b \geqslant 0}$, where $\mathbf{A}$ is an m-by-n matrix with $\mathbf{n \geqslant m}$. Number of all basic feasible solutions to this system is not bigger than $\dfrac{n!}{m!(n-m)!}$. Construct a constraint system with two equations ($\mathbf{m = 2}$) and four variables ($\mathbf{n = 4}$) that has

(i) no basic feasible solutions
(ii) exactly one basic feasible solution
(iii) exactly three distinct basic feasible solutions

Hint: You may wish to perform numerical experiments using function **feassol** discussed in Section 6.4 of this tutorial.

5. Let **X** denote the constraint set that is described in Problem 1(i). Its graph is shown on page 10 of this tutorial. Express point $\mathbf{P} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ as a convex combination of its extreme points. Is this representation unique?
Hint: Compute the extreme points $\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_n}$ of **X** using function **extrpts**. Next form a system of linear equations

$$\lambda_1\mathbf{P_1} + \lambda_2\mathbf{P_2} + \ldots + \lambda_n\mathbf{P_n} = \mathbf{P}$$
$$\lambda_1 + \lambda_2 + \ldots + \lambda_n = 1$$

where all the λ's are nonnegative. Function **feassol** can be used to find all nonnegative solutions to this system.

6. Repeat Problem 5 with $\mathbf{P} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $\mathbf{P} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$.

7. The feasible region described by the inequalities of Problem 1(i) is bounded. An optimal solution to the minimization (maximization) problem can be found by evaluating the objective function at the extreme points of the feasible region and next finding the smallest (largest) value of the objective function. MATLAB has two functions **min** and **max** for computing the smallest and largest numbers in a set. The following command
[z, ind] = min(t) finds the smallest number in the set **t** together with the index **ind** of the element sought. In this exercise you are to find an optimal solution and the corresponding objective value of the Problem 1(i).
Remark. This method for solving the LP problems is **not** recommended in practice because of its high computational complexity. It is included here for pedagogical reasons only.

8. The following LP problem

$$\text{min (max)} = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$
$$\text{Subject to} \quad x_1 + x_2 + \ldots + x_n = 1$$
$$x_1, x_2, \ldots, x_n \geq 0$$

can be solved without using advanced computational methods that are discussed in the Linear Programming class. Write MATLAB function **[x, z] = oneconstr(type, c)** which takes two input parameters **type**, where **type = 'min'** for the minimization problem and **type = 'max'** for the maximization problem and **c** – vector of the cost coefficients and returns the optimal solution **x** and the associated objective value **z**.

9. The following linear programming problem

$$\text{min (max) } z = c*x$$
$$\text{Subject to} \quad Ax \leq b$$
$$x \text{ unrestricted}$$

with $\mathbf{b} \geq \mathbf{0}$ can be solved using function **simplex** (see Section 6.10). A trick is to use a substitution $\mathbf{x = x' - x''}$, where $\mathbf{x', x'' \geq 0}$. Write MATLAB function
[x, z, A] = simplexu(type, c, A, b) which computes an optimal solution to the LP problem that is formulated above. The input and the output parameters have the same meaning as those in function **simplex**. Test your function on the following problem

$$\begin{aligned} \textbf{min } z &= 2x_1 - x_2 \\ \textbf{Subject to } & x_1 + x_2 \leqslant 2 \\ & -x_1 + x_2 \leqslant 1 \\ & x_1 - 2x_2 \leqslant 2 \\ & x_1, x_2 \textbf{ unrestricted} \end{aligned}$$

Also, find an optimal solution to the maximization problem using the same objective function and the same constraint set. Drop the first inequality in the constraint set and solve the maximization problem for the same objective function.

10. In this exercise you are to write MATLAB function **x = nnsol(A, b)** which computes a nonnegative solution **x**, if any, to the system of linear equations **Ax = b**. A trick that can be used is to add a single artificial variable to each equation and next apply a technique of Phase 1 of the Dual Simplex Algorithm with the objective function being equal to the sum of all artificial variables. See [3], Section 3.6 for more details. Recall that function **feassol**, discussed earlier in this tutorial, computes all nonnegative solutions to the constraint system that is in the standard form. A method used in this function is quite expensive and therefore is not recommended for computations with large systems.

11. Write MATLAB function **Y = rotright(X, k)** that takes a matrix **X** and a nonnegative integer **k** and returns a matrix **Y** whose columns are obtained from columns of the matrix **X** by cycling them **k** positions to the right, i.e., if $\mathbf{X = [\, x_1, x_2, \ldots, x_n]}$, then $\mathbf{Y = [x_{k+1}, \ldots, x_n, x_1, \ldots, x_k]}$, where $\mathbf{x_l}$ stands for the lth column of **X**. Function **rotright** may be used by MATLAB functions you are to write in Problems 12 and 13.

12. The following problem is of interest in interpolation by shape preserving spline functions. Given nonnegative numbers $\mathbf{b_1, b_2, \ldots, b_n}$. Find the nonnegative numbers $\mathbf{p_1, p_2, \ldots, p_{n+1}}$ that satisfy a system of linear inequalities

$$\mathbf{2p_l + p_{l+1} \leqslant b_l \leqslant p_l + 2p_{l+1}, \quad l = 1, 2, \ldots, n.}$$

This problem can be converted easily to the problem of finding a basic feasible solution of a constraint set in standard form **Ap = d, p ⩾ 0**.

(i) What is the structure of the constraint matrix **A**?
(ii) How would you compute the column vector **d**?
(iii) Write MATLAB function **p = slopec(b)** which computes a nonnegative vector **p** that satisfies the system of inequalities in this problem. You may wish to make calls from within your function to the function **nnsol** of Problem 10.
(iv) Test your function for the following vectors **b = [1; 2]** and **b = [2; 1]**.

13. Another problem that arose in interpolation theory has the same constraint set as the one in Problem 12 with additional conditions imposed on the legitimate variables $\mathbf{p_1 \leqslant p_2 \leqslant \ldots \leqslant p_{n+1}}$. Answer the questions (i) and (ii) of Problem 12 and write MATLAB function **p = slopei(b)** which computes a solution to the associated LP problem. Also, test your function using vectors **b = [1; 2; 3]** and **b = [3; 2; 1]**.

14. Some LP problems can be solved using different methods that are discussed in this tutorial. A choice of a right method is often based on the criterion of its computational complexity. MATLAB allows a user to count a number of flops (the floating point operations +, -, *, /) needed by a particular algorithm. To obtain an information about a number of flops used type the following commands:

    flops(0)
    functionname
    flops

    First command sets up the counter to zero. Next a function is executed and finally a number of flops used is displayed in the **Command Window**.
    Suppose that one wants to find an optimal solution to the following LP problem

    $$\begin{aligned} &\textbf{max (min) } z = c^*x \\ &\textbf{Subject to } Ax \leqslant b \\ &x \geqslant 0 \end{aligned}$$

    ($\mathbf{b \geqslant 0}$). Function **simplex**, included in Section 6.10, can be used to compute an optimal solution of this problem. Another option is to use function **simplex2p** (see Section 6.7). Third method that would be used is the Dual Simplex Algorithm discussed in Section 6.8. Compare computational complexity of these three methods solving the LP problems of your choice. You may begin your experiment using the LP problem in Example 1 (see [3], page 104).

15. Time of execution of a function is also considered as a decisive factor in choosing a computational method for solving a particular problem. MATLAB has a pair of functions named **tic** and **toc** used to measure a time of computations. To use these functions issue the following commands: **tic, functionname; toc**. Compare time of execution of three functions **simplex**, **simplex2p**, and **dsimplex** solving the LP problems of your choice. During the execution of functions **simplex2p** and **dsimplex** you will be asked whether or not you wish to monitor the progress of computations. In both instances click on the **No** button.

16. Consider the following LP problem with bounded variables

    $$\begin{aligned} &\textbf{min (max) } z = c^T x \\ &\textbf{Subject to } Ax \leqslant b \\ &0 \leqslant x \leqslant u \end{aligned}$$

    where **u** is a given positive vector and $\mathbf{b \geqslant 0}$. Special methods for solving this problem are discussed in [1] and [2]. For the LP problems with a small number of constraints one can add the upper bound constraints $\mathbf{x \leqslant u}$ to the original constraint set and next solve the new problem using the simplex method. In this exercise you are to write MATLAB function **[x, z, B] = simplub(type, c, A, b, u)** which computes an optimal solution **x** to the problem under discussion. Other output parameters include the objective value **z** at the optimal solution and the final tableau **B** that is generated by the simplex method. You may wish to use the function **simplex** included in Section 6.10. Test your function on the LP problems of your choice.

17. Use function **simplex2p** to solve the optimization problem discussed in [3], Appendix C, pp. 375 - 376. Explain why this problem is degenerated. Display all the intermediate tableaux generated by the function **simplex2p**.

18. Run function **cpa** on the LP problem that is discussed in [3], Example 3, pp. 208-209 (see also pages 44 - 45 of this tutorial). Analyze the intermediate tableaux generated by this function. Explain why function **cpa** terminates computations without finding an optimal solution.