IE 535 – LINEAR PROGRAMMING PROJECT

Name: **ANESH KRISHNA J N**

Puid : **30142658**

Software used : **MATLAB**

Models Solved: **MODEL 25* and MODEL 26**

Model- 26:

Decision Variables:

Let:

x - fractional participation in the Foster City problem

y - fractional participation in Lower-Middle

z – Disney participation

b_i – amount borrowed in period i in **Millions** of dollars, i = 1,2,3,4,5,6

l_i – amount lent in period i in **Millions** of dollars. i = 1,2,3,4,5,6.

W_{net} – Net worth of WSDM at the end of three years

Objective Function:

Max = W_{net}

Subject to:

$3 * x + 2 * y + 2 * z - b_1 + l_1 = 2;$

$1 * x + 0.5 * y + 2 * z + 1.035 * b_1 - 1.03 * l_1 - b_2 + l_2 = 0.5;$

$1.8 * x - 1.5 * y + 1.8 * z + 1.035 * b_2 - 1.03 * l_2 - b_3 + l_3 = 0.4;$

$-0.4 * x - 1.5 * y - 1 * z + 1.035 * b_3 - 1.03 * l_3 - b_4 + l_4 = 0.38;$

$-1.8 * x - 1.5 * y - 1 * z + 1.035 * b_4 - 1.03 * l_4 - b_5 + l_5 = 0.36;$

$-1.8 * x - 0.2 * y - 1 * z + 1.035 * b_5 - 1.03 * l_5 - b_6 + l_6 = 0.34;$

$W_{net} - 5.5 * x + 1 * y - 6 * z + 1.035 * b_6 - 1.03 * l_6 = 0.3;$

$b_1 <= 2$

$b_2 <= 2$

$b_3 <= 2$

$b_4 <= 2$

$b_5 <= 2$

b_6 <= 2

x <= 1

y <= 1

z <= 1

CODE:

A (before converting to standard form), b and c are the inputs (c is given in the name of cc)

```
% Code:
%% Stage 1: To convert the Problem into standard form and also to add
artificial variables if necessary PLUS REDUNDANCY REMOVED

% coefficient matrix in its raw form is given as input:

A = [

         0    3.0000    2.0000    2.0000   -1.0000         0         0
0         0         0    1.0000         0         0         0         0
0
         0    1.0000    0.5000    2.0000    1.0350   -1.0000         0
0         0         0   -1.0300    1.0000         0         0         0
0
         0    1.8000   -1.5000    1.8000         0    1.0350   -1.0000
0         0         0         0   -1.0300    1.0000         0         0
0
         0   -0.4000   -1.5000   -1.0000         0         0    1.0350   -
1.0000         0         0         0         0   -1.0300    1.0000
0         0
         0   -1.8000   -1.5000   -1.0000         0         0         0
1.0350   -1.0000         0         0         0         0   -1.0300
1.0000         0
         0   -1.8000   -0.2000   -1.0000         0         0         0
0    1.0350   -1.0000         0         0         0         0   -1.0300
1.0000
    1.0000   -5.5000    1.0000   -6.0000         0         0         0
0         0    1.0350         0         0         0         0         0   -
1.0300
         0         0         0         0    1.0000         0         0
0         0         0         0         0         0         0         0
0
         0         0         0         0         0    1.0000         0
0         0         0         0         0         0         0         0
0
         0         0         0         0         0         0    1.0000
0         0         0         0         0         0         0         0
0
         0         0         0         0         0         0         0
1.0000         0         0         0         0         0         0
0         0
         0         0         0         0         0         0         0
0    1.0000         0         0         0         0         0         0
0
         0         0         0         0         0         0         0
0         0    1.0000         0         0         0         0         0
0
```

```
         0    1.0000         0         0         0         0         0
0         0         0         0         0         0         0         0
0
         0         0    1.0000         0         0         0         0
0         0         0         0         0         0         0         0
0
         0         0         0    1.0000         0         0         0
0         0         0         0         0         0         0         0
0
]

v = [0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1]                            % A vector
'v' holding depicting the signs of the constraints;

% (-1 for >=), (0 for =) and (1 for <=)
[m n] = size(A);             % this will return # of rows to 'm' and # of
columns to 'n'

mat = zeros(m,1);            % this will create a zero vector of dim m*1


for k=1:m
if v(k)== 0
mat(k) = 1;                     % If the sign is '=', it adds an artificial
variable; this is done by adding column
                                %with a an entry = 1 corresponding to that
constraint
A = [A mat];
mat = zeros(m,1);
end
end


for k=1:m
if v(k)== -1
mat(k) = 1;                     % If the sign is '>=' it addds an
artificial variable
A = [A mat];                    %with a an entry = 1 corresponding to that
constraint
mat = zeros(m,1);
end
end

[ba ab] = size(A)          % to define cost vector for the first phase, we
will be using
                           %this as all the artificial variables required have
been added.

for k=1:m
if v(k)== 1
mat(k) = 1;
A = [A mat];               % if the sign is '<=' it adds a column with an
entry = 1 corresponding to that constraint.
mat = zeros(m,1);          %Thus a slack variable is added.
end
end

[bbb ccc] = size(A)          %to define indices of columns in initial basis
we wil be using this;
```

```
                                % Now, we have created an initial Basis which
                                % is identity


for k=1:m
if v(k)== -1
mat(k) = -1;            % if the sign is '<=' it adds a column with an
entry = -1 corresponding to that constraint.
A = [A mat];            %Thus a slack variable is added.
mat = zeros(m,1);
end
end



A;                              % We now have a matrix A that is ready to be
used in phase 1

% Redundancy Check : checking condition included at the last
[row col] = size(A)
B = A
for i = 1 : row
for j = 1 : col
B(i,j) = Inf;                               % We are definig a matrix B of the
dimensions same as A with entries as INFINITY
end
end
count_red = 0                               % We define an arbitrary variable
cout_red as ZERO
for t = 1 : row
    for r = t+1 : row
    u = zeros(col,1);                        % We are also defining a zero
vector which will hold the RATIOS of values
                                            % present in two rows that are
being checked
    for p = 1:col
    u(p,1) = (A(r,p)/A(t,p)) ;
    end
    k = u(1)
        for v = 1 : col
        if u(v) == k
        count_red = count_red + 1;           % We are checking if all the
ratios are same
        end
        end


        if count_red == col
            if k >=1
                for w = 1 : col
                B(r,w) = 0  ;                % If the ratio > 1, it means the
row with higher index is redundant
                                            % If the ratios are same
corresponding row in B is assigned ZERO
                B;
                count_red=0;
                end
            else
                for w = 1 : col
                B(t,w) = 0 ;                 % If the ratio > 1, it means the
row with lower index is redundant
                                            % If the ratios are same
corresponding row in B is assigned ZERO
```

```
                B;
                count_red=0;
                end
            end
        end
        count_red = 0;
    end
end


% Checkig condition : if any particular row is zero in B matrix,
corresponding row in A is redundant
z = zeros(row, 1)
for i = 1 : row
if B(i,:)==0            % We check if any of the row in B has all entries
as ZERO; If so corresponding row in A is redundant
z(i) = i
end
end

for i = 1 : row
if z(i) > 0
    A(z(i),:) = []      % The row in A that corresponds to the row in B
that has all zeros, is finally removed here.
    z(i) = 0
    z = z - 1
end
end
A
% End of redundancy check

[mm nn] = size(A);


b = [2;.5;.4;.38;.36;.34;.3;2;2;2;2;2;2;1;1;1];         % b vector and cost
vector, defined by 'cc' are given as inputs

cc = [-1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

c = zeros(nn,1);
c(n+1 : ab) = 1;

iB = [n+1 : ccc]        % We are defining the the columns that should enter
the initial Basis

tt = iB(1) - 1;         % This will be used later to remove constraints
from A that are redundant at the end of phase 1.
mB = A(:, iB);          % Initial Basis is extracted from A
[row_count col_count] = size(A);

%% Stage 2
enter = 5;              % We assign enter to be the maximum value of
reduced cost coefficient(denoted by rcc) at a later stage
                        % If enter >0, it goes to the next iteration; else
                        % it returns the optimal solution. This is
                        % implementd here with while. For the first
                        % iteration to be started, we give an arbitrary
                        % value to 'enter'
while enter > 0.000000001
```

```
x = zeros(col_count,1);
x(iB) = inv(mB)*b;          % BFS is calculated




rcc = c(iB)'*inv(mB)*A-c' ;    % rcc is reduced cost coeffiecient and it is
calculated in this step

enter = max(rcc);              % enter is given max value present in rcc

if enter < 0.000000001
fprintf('current basis is optimal');    % Optimality check: It enter < 0,
it retrns the optimal solution
                                         % and optimal objective
value(Zopt); Else the simplex continues
x
Zopt = c(iB)'*inv(mB)*b
Zopt
break
end



%% Stage 3: Checking if theproblem is degenerate
deg_check = x(iB);
[~,basis_variable_count] = size(iB);
count = 1;                               % We check if any of Basis
variables is zero
for s = 1 : basis_variable_count         % We define an arbitrary variable
(count) as 1 before starting the degenracy check
    if(deg_check(s) == 0)                % if there is any Basis variables
that is zero, count will be incremented by 1
                                         % If count > 1 we say it's
                                         % degenerate.
    count = count + 1;
    end
end
count


% If the problem is degenerate, the code uses Bland's rule to select a
variable to enter the basis as per the following:
if count > 1
for u = 1 : col_count
    if rcc(u) > 0.00000001
    entering_index = u;      % We are finding the first index of RCC that is
greater than zero.
    enter = rcc(u)
    break
    end
end
else
for t = 1 : col_count
    if rcc(t)== enter
    entering_index = t;        % If not degenerate, normal simplex is
happening here
```

```
    break
    end
end
end

% End of this degeneracy code and simplex continues



%% Stage 4 : We check whether the LP is bounded or not.

ubc = zeros(col_count,1);                           % ubc is unbounded check

ubc(iB) = inv(mB)*A(:,entering_index);

var = max(ubc);                   % We assign max value of ubc to var

if var < 0.000000001              % if var < 0, it's unbounded;
fprintf('LP is unbounded');
break
end


for e= 1:col_count
        if ubc(e) <= 0.0000001                     % If the Lp is bounded we
need to find the ratio for all POSITIVE values of ubc
                                                   % and to choose the minimum
                                                   % of that.. So assign all
                                                   % NON POSITIVE values to
                                                   % INFINITY in ubc
        ubc(e) = Inf;
        end
end
                                                    % here we the ratio and
                                                    % also the minimum of that.
for d = 1:col_count
if ubc(d) > 0.000000001 && ubc(d) ~= inf
ubc(d) = x(d)/ubc(d);
end
end

leave= min(ubc);

%% Stage 5 If the problem is degenerate, the code uses Bland's rule to
select a variable to leavethe basis as per the following:
if count > 1
for r = 1 : col_count
    if ubc(r)== leave
                                        % if there is degeneracy leave = 0
    exiting_index= r;                   % We are finding the first index of
ubc whose rato is zero is greater than zero.
    break
    end
end
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);
    iB(t) = entering_index;             % Here the index of entering column
replaces the index of leaving column in Basis
```

```
    break
    end
end
else
[leave exiting_index] = min(ubc);
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);                              % Else normal simplex is happening
here and index of entering column
                                            % replaces the index of leaving
column in Basis
    iB(t) = entering_index;
    break
    end
end
end

% End of this degeneracy code and simplex continues


iB;
mB = A(:, iB);

end



%% Stage 6: PHASE 1 to PHASE 2

if Zopt > 0
fprintf('The given LP is infeasible\n')   % At the end of Phase 1, we are
checking if LP s feasible

else
% if the LP is feasible, we start the building of start of Phase 2
count_iB = 0
z = zeros(col_count,1);
for j = 1 : col_count
if c(j) == 1
z(j) = j;
end
end
                                    % These two for loops serve ONE
purpose. That is to remove the columns corresponding to
                                    % to artificial variables in 'A'
for i = 1 : col_count
if z(i) > 0
    A(:,z(i)) = [];
    z(i) = 0;
    z = z - 1;
    count_iB = count_iB + 1;
end
end




v = zeros(col_count,1);
for j = 1 : col_count
if c(j) == 1
v(j) = j;
```

```
end
end
                                    % These two for loops serve ONE
purpose. That is to remove the COLUMNS corresponding to
                                    % to artificial variables in 'C'
for i = 1 : col_count
if v(i) > 0
    c(v(i),:) = [];
    v(i) = 0;
    v= v - 1;
end
end

A
b
c = cc




for t = 1: row_count
if iB(t) > tt && iB(t) <= count_iB + tt     % We are checking if any of the
artificial variable is present in Optimal solution of PHASE 1
                                            % If so, corresponding row in A
                                            % is redundant and hence we are
                                            % removing the constraint as as
                                            % whole.
A(t,:) = [];
iB(t) = [];
b(t) = [];
break
end
end

[row_count col_count] = size(A);
for t = 1: row_count
if iB(t) > tt                               % In the above discussed
scenario we have to remove the column corresponding to the
                                            % artificial variable that is
                                            % present in Basis.
iB(t) = iB(t) - count_iB;
end
end
iB;
A;
b;
c;




%% Stage 7: Phase 2
% We are finding if the solution at the end of phase 1 after removing all
artificial variables is optimal or not. IF not optimal, we proceed with 2nd
phase.

mB = A(:, iB);
[row_count col_count] = size(A);

x = zeros(col_count,1);
x(iB) = inv(mB)*b
```

```
rcc = c(iB)'*inv(mB)*A-c'
%rcc is reduced cost coeffiecient

enter = max(rcc)

if enter < 0.000000001
fprintf('current basis is optimal');
x
Zopt = c(iB)'*inv(mB)*b
Zopt

else
while enter > 0.000000001

x = zeros(col_count,1);
x(iB) = inv(mB)*b;                    % BFS is calculated




rcc = c(iB)'*inv(mB)*A-c' ;           % rcc is reduced cost coeffiecient
and it is calculated in this step

enter = max(rcc);                     % enter is given max value present
in rcc

if enter < 0.000000001
fprintf('current basis is optimal');    % Optimality check: It enter < 0,
it retrns the optimal solution
                                        % and optimal objective
value(Zopt); Else the simplex continues
x
Zopt = c(iB)'*inv(mB)*b
Zopt
fprintf('Therefore, Maximised WSDMs net worth is %d \n', (-1 *Zopt))
break
end



%% Stage 8: Checking if theproblem is degenerate
deg_check = x(iB);
[~,basis_variable_count] = size(iB);
count = 1;                            % We check if any of Basis
variables is zero
for s = 1 : basis_variable_count      % We define an arbitrary variable
(count) as 1 before starting the degenracy check
    if(deg_check(s) == 0)             % if there is any Basis variables
that is zero, count will be incremented by 1
                                      % If count > 1 we say it's
                                      % degenerate.

    count = count + 1;
    end
end
count
```

```
% If the problem is degenerate, the code uses Bland's rule to select a
variable to enter the basis as per the following:
if count > 1
for u = 1 : col_count
    if rcc(u) > 0.00000001
    entering_index = u;                              % We are finding the first
index of RCC that is greater than zero.
    enter = rcc(u)
    break
    end
end
else
for t = 1 : col_count
    if rcc(t)== enter
    entering_index = t;                              % If not degenerate, normal
simplex is happening here
    break
    end
end
end

% End of this degeneracy code and simplex continues



%% Stage 9 : We check whether the LP is bounded or not.

ubc = zeros(col_count,1);                           % ubc is unbounded check

ubc(iB) = inv(mB)*A(:,entering_index);

var = max(ubc);                                     % We assign max value of ubc
to var

if var < 0.000000001                                % if var < 0, it's
unbounded;
fprintf('LP is unbounded');
break
end


for e= 1:col_count
        if ubc(e) <= 0.0000001                      % If the Lp is bounded we
need to find the ratio for all POSITIVE values of ubc
                                                    % and to choose the minimum
                                                    % of that.. So assign all
                                                    % NON POSITIVE values to
                                                    % INFINITY in ubc
        ubc(e) = Inf;
        end
end
                                                     % here we the ratio and
                                                     % also the minimum of that.
for d = 1:col_count
if ubc(d) > 0.000000001 && ubc(d) ~= inf
ubc(d) = x(d)/ubc(d);
end
end
```

```
leave= min(ubc);

%% Stage 10: If the problem is degenerate, the code uses Bland's rule to
select a variable to leavethe basis as per the following:
if count > 1
for r = 1 : col_count
    if ubc(r)== leave
                                        % if there is degeneracy leave = 0
    exiting_index= r;                   % We are finding the first index of
ubc whose rato is zero is greater than zero.
    break
    end
end
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);
    iB(t) = entering_index;             % Here the index of entering column
replaces the index of leaving column in Basis
    break
    end
end
else
[leave exiting_index] = min(ubc);
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);                          % Else normal simplex is happening
here and index of entering column
                                        % replaces the index of leaving
column in Basis
    iB(t) = entering_index;
    break
    end
end
end

% End of this degeneracy code and simplex continues


iB;
mB = A(:, iB);

end



end                                     % for the if just above second
while

end                                     % for the if to move from phase 1
to phase 2
```

OUTPUTS:

1. **At the end of Phase 1**

```
current basis is optimal
x =

    7.0415
         0
    0.1963
    0.8037
         0
    1.2056
    2.0000
    0.5919
         0
         0
         0
         0
         0
         0
    0.8456
    2.0539
         0
         0
         0
         0
         0
         0
         0
    2.0000
    0.7944
         0
    1.4081
    2.0000
    2.0000
    1.0000
    0.8037
    0.1963
```

```
Zopt =

     0

The given LP is feasible
```

2. **At the end of phase 2**

```
current basis is optimal
x =

    7.6652
    0.7143
    0.6372
         0
    1.4174
    2.0000
    2.0000
    0.4484
         0
         0
         0
         0
         0
         0
    2.1375
    3.9549
    0.5826
         0
         0
    1.5516
    2.0000
    2.0000
    0.2857
    0.3628
    1.0000
```

```
Zopt =

   -7.6652

Therefore, Maximised WSDMs net worth is 7.665179e+00
```

Commercial Solver Used is MATLAB:

Inputs:

```
A =

     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0

Trial>> b

b =

     2
     2
     2
     2
     2
     2
     1
     1
     1
```

```
Aeq =

         0    3.0000    2.0000    2.0000   -1.0000         0         0         0         0         0    1.0000         0         0         0         0         0
         0    1.0000    0.5000    2.0000    1.0350   -1.0000         0         0         0         0   -1.0300    1.0000         0         0         0         0
         0    1.8000   -1.5000    1.8000         0    1.0350   -1.0000         0         0         0         0   -1.0300    1.0000         0         0         0
         0   -0.4000   -1.5000   -1.0000         0         0    1.0350   -1.0000         0         0         0         0   -1.0300    1.0000         0         0
         0   -1.8000   -1.5000   -1.0000         0         0         0    1.0350   -1.0000         0         0         0         0   -1.0300    1.0000         0
         0   -1.8000   -0.2000   -1.0000         0         0         0         0    1.0350   -1.0000         0         0         0         0   -1.0300    1.0000
    1.0000   -5.5000    1.0000   -6.0000         0         0         0         0         0    1.0350         0         0         0         0         0   -1.0300

Trial>> Beq

Beq =

    2.0000
    0.5000
    0.4000
    0.3800
    0.3600
    0.3400
    0.3000
```

```
f =

    -1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

Trial>> lb

lb =

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

Trial>> ub

ub =

     []
```

```
Trial>> [x z] = linprog(f,A,b,Aeq,Beq,lb,ub)

Optimal solution found.


x =

    7.6652
    0.7143
    0.6372
         0
    1.4174
    2.0000
    2.0000
    0.4484
         0
         0
         0
         0
         0
         0
    2.1375
    3.9549


z =

   -7.6652
```

Hence, the optimal objective value obtained in both the code and commercial solver match even though the optimal solution is different.

Model-25*:
Decision Variables:

Let,
For Amides:
Pai be number of units produced in period i, for i = 1, 2, 3, and 4;
Iai be units in inventory at the end of period i; i = 1, 2, 3, and 4;
Uai = increase in production level between period i – 1 and i; i = 1, 2, 3, and 4;
Dai = decrease in production level between i – 1 and i. i = 1, 2, 3, and 4;

For Nitrile,
Pni be number of units produced in period i, for i = 1, 2, 3, and 4; i = 1, 2, 3, and 4;
Ini be units in inventory at the end of period i; i = 1, 2, 3, and 4;
Uni = increase in production level between period i – 1 and i; i = 1, 2, 3, and 4;
Dni = decrease in production level between i – 1 and i. i = 1, 2, 3, and 4;

For Amides,
Rai be regular working hours for each period i, i = 1, 2, 3, and 4;
Oai be overtime for each period i, i = 1, 2, 3, and 4;

For Nitrile,
Rni be regular working hours for each period i, i = 1, 2, 3, and 4;
Oni be overtime for each period i, i = 1, 2, 3, and 4;

Objective Function:
MIN :
8 * Ia1 + 8 * Ia2 + 8 * Ia3 + 8 * Ia4 + 7 * In1 + 7 * In2 + 7 * In3 + 7 * In4 + 11 * Ua1 + 11 * Ua2 + 11 * Ua3 + 11 * Ua4 +11 * Ua5+ 11 * Da1 + 11 * Da2 + 11 * Da3 + 11 * Da4 + 11 * Da5 + 11 * Un1 + 11 * Un2 + 11 * Un3 + 11 * Un4 +11 * Un5+ 11 * Dn1 + 11 * Dn2 + 11 * Dn3 + 11 * Dn4 + 11 * Dn5 + 110 * ra1 + 160 * oa1 + 110 * ra2 + 160 * oa2 + 110 * ra3 + 160 * oa3 + 110 * ra4 + 160 * oa4 + 135 * rn1 + 190 * rn1 + 135 * rn2 + 190 * rn2+ 135 * rn3+ 190 * rn3+ 135 * rn4 + 190 * rn4

Constraints:

Pa1 = 20 + Ia1;
Ia1 + Pa2 = 30 + Ia2;
Ia2 + Pa3 = 50 + Ia3;
Ia3 + Pa4 = 60 + Ia4;

Ua1 - Da1 = Pa1 - 40;
Ua2 - Da2 = Pa2 - Pa1;
Ua3 - Da3 = Pa3 - Pa2;
Ua4 - Da4 = Pa4 - Pa3;
Ua5 - Da5 = 40 - Pa4;

Pn1 = 20 + In1;
In1 + Pn2 = 30 + In2;
In2 + Pn3 = 50 + In3;
In3 + Pn4 = 60 + In4;

Un1 - Dn1 = Pn1 - 40;
Un2 - Dn2 = Pn2 - Pn1;
Un3 - Dn3 = Pn3 - Pn2;
Un4 - Dn4 = Pn4 - Pn3;
U5 - Dn5 = 40 - Pn4;

2oa1 <= ra1
2oa2 <= ra2
2oa3 <= ra3
2oa4 <= ra4
2on1 <= rn1
2on2 <= rn2
2on3 <= rn3
2on4 <= rn4

CODE:

A (before converting to standard form), b and c are the inputs (c is given in the name of cc)

```
% Code:
%% Stage 1: To convert the Problem into standard form and also to add
artificial variables if necessary PLUS REDUNDANCY REMOVED

% coefficient matrix in its raw form is given as input:

A = [-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0;
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 2 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 0 2 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -
1 0 0 0 2 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1 0 0 0 2 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 -1 0 0 0 2 0 0 0 0 0 0 0 0;]


v = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
% A vector 'v' holding depicting the signs of the constraints;

% (-1 for >=), (0 for =) and (1 for <=)
[m n] = size(A);              % this will return # of rows to 'm' and # of
columns to 'n'

mat = zeros(m,1);             % this will create a zero vector of dim m*1


for k=1:m
if v(k)== 0
mat(k) = 1;                       % If the sign is '=', it adds an artificial
variable; this is done by adding column
                                  %with a an entry = 1 corresponding to that
constraint
A = [A mat];
mat = zeros(m,1);
end
end


for k=1:m
if v(k)== -1
mat(k) = 1;                       % If the sign is '>=' it addds an
artificial variable
A = [A mat];                      %with a an entry = 1 corresponding to that
constraint
mat = zeros(m,1);
end
end

[ba ab] = size(A)         % to define cost vector for the first phase, we
will be using
                          %this as all the artificial variables required have
been added.
```

```
for k=1:m
if v(k)== 1
mat(k) = 1;
A = [A mat];             % if the sign is '<=' it adds a column with an
entry = 1 corresponding to that constraint.
mat = zeros(m,1);        %Thus a slack variable is added.
end
end

[bbb ccc] = size(A)          %to define indices of columns in initial basis
we wil be using this;
                             % Now, we have created an initial Basis which
                             % is identity


for k=1:m
if v(k)== -1
mat(k) = -1;             % if the sign is '<=' it adds a column with an
entry = -1 corresponding to that constraint.
A = [A mat];             %Thus a slack variable is added.
mat = zeros(m,1);
end
end



A;                           % We now have a matrix A that is ready to be
used in phase 1

% Redundancy Check : checking condition included at the last
[row col] = size(A)
B = A
for i = 1 : row
for j = 1 : col
B(i,j) = Inf;                            % We are definig a matrix B of the
dimensions same as A with entries as INFINITY
end
end
count_red = 0                            % We define an arbitrary variable
cout_red as ZERO
for t = 1 : row
    for r = t+1 : row
    u = zeros(col,1);                     % We are also defining a zero
vector which will hold the RATIOS of values
                                         % present in two rows that are
being checked
    for p = 1:col
    u(p,1) = (A(r,p)/A(t,p)) ;
    end
    k = u(1)
        for v = 1 : col
        if u(v) == k
        count_red = count_red + 1;        % We are checking if all the
ratios are same
        end
        end

        if count_red == col
            if k >=1
                for w = 1 : col
```

```
                B(r,w) = 0  ;             % If the ratio > 1, it means the
row with higher index is redundant
                                         % If the ratios are same
corresponding row in B is assigned ZERO
                B;
                count_red=0;
                end
            else
                for w = 1 : col
                B(t,w) = 0 ;              % If the ratio > 1, it means the
row with lower index is redundant
                                         % If the ratios are same
corresponding row in B is assigned ZERO
                B;
                count_red=0;
                end
            end
        end
        count_red = 0;
    end
end


% Checkig condition : if any particular row is zero in B matrix,
corresponding row in A is redundant
z = zeros(row, 1)
for i = 1 : row
if B(i,:)==0             % We check if any of the row in B has all entries
as ZERO; If so corresponding row in A is redundant
z(i) = i
end
end

for i = 1 : row
if z(i) > 0
    A(z(i),:) = []       % The row in A that corresponds to the row in B
that has all zeros, is finally removed here.
    z(i) = 0
    z = z - 1
end
end
A
% End of redundancy check

[mm nn] = size(A);


b = [20;30;50;60;20;50;30;60;40;0;0;0;40;40;0;0;0;40;0;0;0;0;0;0;0;0];
% b vector and cost vector, defined by 'cc' are given as inputs

cc =
[8;8;8;8;7;7;7;7;11;11;11;11;11;11;11;11;11;11;11;11;11;11;11;11;11;11;11;1
1;110;160;110;160;110;160;110;160;135;195;135;195;135;195;135;195;0;0;0;0;0
;0;0;0;0;0;0;0;0;0;0;0];

c = zeros(nn,1);
c(n+1 : ab) = 1;
```

```
iB = [n+1 : ccc]        % We are defining the the columns that should enter
the initial Basis

tt = iB(1) - 1;         % This will be used later to remove constraints
from A that are redundant at the end of phase 1.
mB = A(:, iB);          % Initial Basis is extracted from A
[row_count col_count] = size(A);

%% Stage 2
enter = 5;              % We assign enter to be the maximum value of
reduced cost coefficient(denoted by rcc) at a later stage
                        % If enter >0, it goes to the next iteration; else
                        % it returns the optimal solution. This is
                        % implemented here with while. For the first
                        % iteration to be started, we give an arbitrary
                        % value to 'enter'
while enter > 0.000000001

x = zeros(col_count,1);
x(iB) = inv(mB)*b;          % BFS is calculated





rcc = c(iB)'*inv(mB)*A-c' ;    % rcc is reduced cost coeffiecient and it is
calculated in this step

enter = max(rcc);              % enter is given max value present in rcc

if enter < 0.000000001
fprintf('current basis is optimal');    % Optimality check: It enter < 0,
it retrns the optimal solution
                                        % and optimal objective
value(Zopt); Else the simplex continues
x
Zopt = c(iB)'*inv(mB)*b
Zopt
break
end




%% Stage 3: Checking if theproblem is degenerate
deg_check = x(iB);
[~,basis_variable_count] = size(iB);
count = 1;                              % We check if any of Basis
variables is zero
for s = 1 : basis_variable_count        % We define an arbitrary variable
(count) as 1 before starting the degenracy check
    if(deg_check(s) == 0)               % if there is any Basis variables
that is zero, count will be incremented by 1
                                        % If count > 1 we say it's
                                        % degenerate.

    count = count + 1;
    end
end
count
```

```
% If the problem is degenerate, the code uses Bland's rule to select a
variable to enter the basis as per the following:
if count > 1
for u = 1 : col_count
    if rcc(u) > 0.00000001
    entering_index = u;     % We are finding the first index of RCC that is
greater than zero.
    enter = rcc(u)
    break
    end
end
else
for t = 1 : col_count
    if rcc(t)== enter
    entering_index = t;        % If not degenerate, normal simplex is
happening here
    break
    end
end
end

% End of this degeneracy code and simplex continues



%% Stage 4 : We check whether the LP is bounded or not.

ubc = zeros(col_count,1);                           % ubc is unbounded check

ubc(iB) = inv(mB)*A(:,entering_index);

var = max(ubc);                     % We assign max value of ubc to var

if var < 0.000000001                % if var < 0, it's unbounded;
fprintf('LP is unbounded');
break
end



for e= 1:col_count
        if ubc(e) <= 0.0000001                  % If the Lp is bounded we
need to find the ratio for all POSITIVE values of ubc
                                                % and to choose the minimum
                                                % of that.. So assign all
                                                % NON POSITIVE values to
                                                % INFINITY in ubc
        ubc(e) = Inf;
        end
end
                                                 % here we the ratio and
                                                 % also the minimum of that.
for d = 1:col_count
if ubc(d) > 0.000000001 && ubc(d) ~= inf
ubc(d) = x(d)/ubc(d);
end
end
```

```
leave= min(ubc);

%% Stage 5 If the problem is degenerate, the code uses Bland's rule to
select a variable to leavethe basis as per the following:
if count > 1
for r = 1 : col_count
    if ubc(r)== leave
                                        % if there is degeneracy leave = 0
    exiting_index= r;                   % We are finding the first index of
ubc whose rato is zero is greater than zero.
    break
    end
end
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);
    iB(t) = entering_index;             % Here the index of entering column
replaces the index of leaving column in Basis
    break
    end
end
else
[leave exiting_index] = min(ubc);
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);                          % Else normal simplex is happening
here and index of entering column
                                        % replaces the index of leaving
column in Basis
    iB(t) = entering_index;
    break
    end
end
end

% End of this degeneracy code and simplex continues


iB;
mB = A(:, iB);

end



%% Stage 6: PHASE 1 to PHASE 2

if Zopt > 0
fprintf('The given LP is infeasible\n')    % At the end of Phase 1, we are
checking if LP s feasible

else
% if the LP is feasible, we start the building of start of Phase 2
count_iB = 0
z = zeros(col_count,1);
for j = 1 : col_count
if c(j) == 1
z(j) = j;
end
end
```

```
                                    % These two for loops serve ONE
purpose. That is to remove the columns corresponding to
                                    % to artificial variables in 'A'
for i = 1 : col_count
if z(i) > 0
    A(:,z(i)) = [];
    z(i) = 0;
    z = z - 1;
    count_iB = count_iB + 1;
end
end



v = zeros(col_count,1);
for j = 1 : col_count
if c(j) == 1
v(j) = j;
end
end
                                    % These two for loops serve ONE
purpose. That is to remove the COLUMNS corresponding to
                                    % to artificial variables in 'C'
for i = 1 : col_count
if v(i) > 0
    c(v(i),:) = [];
    v(i) = 0;
    v= v - 1;
end
end

A
b
c = cc



for t = 1: row_count
if iB(t) > tt && iB(t) <= count_iB + tt      % We are checking if any of the
artificial variable is present in Optimal solution of PHASE 1
                                             % If so, corresponding row in A
                                             % is redundant and hence we are
                                             % removing the constraint as as
                                             % whole.
A(t,:) = [];
iB(t) = [];
b(t) = [];
break
end
end

[row_count col_count] = size(A);
for t = 1: row_count
if iB(t) > tt                                % In the above discussed
scenario we have to remove the column corresponding to the
                                             % artificial variable that is
                                             % present in Basis.
iB(t) = iB(t) - count_iB;
end
end
```

```
iB;
A;
b;
c;



%% Stage 7: Phase 2
% We are finding if the solution at the end of phase 1 after removing all
artificial variables is optimal or not. IF not optimal, we proceed with 2nd
phase.

mB = A(:, iB);
[row_count col_count] = size(A);

x = zeros(col_count,1);
x(iB) = inv(mB)*b
rcc = c(iB)'*inv(mB)*A-c'
%rcc is reduced cost coeffiecient

enter = max(rcc)

if enter < 0.000000001
fprintf('current basis is optimal');
x
Zopt = c(iB)'*inv(mB)*b
Zopt

else
while enter > 0.000000001

x = zeros(col_count,1);
x(iB) = inv(mB)*b;                      % BFS is calculated





rcc = c(iB)'*inv(mB)*A-c' ;             % rcc is reduced cost coeffiecient
and it is calculated in this step

enter = max(rcc);                       % enter is given max value present
in rcc

if enter < 0.000000001
fprintf('current basis is optimal');    % Optimality check: It enter < 0,
it retrns the optimal solution
                                        % and optimal objective
value(Zopt); Else the simplex continues
x
Zopt = c(iB)'*inv(mB)*b
Zopt
break
end
```

```
%% Stage 8: Checking if theproblem is degenerate
deg_check = x(iB);
[~,basis_variable_count] = size(iB);
count = 1;                              % We check if any of Basis
variables is zero
for s = 1 : basis_variable_count        % We define an arbitrary variable
(count) as 1 before starting the degenracy check
    if(deg_check(s) == 0)               % if there is any Basis variables
that is zero, count will be incremented by 1
                                        % If count > 1 we say it's
                                        % degenerate.
    count = count + 1;
    end
end
count


% If the problem is degenerate, the code uses Bland's rule to select a
variable to enter the basis as per the following:
if count > 1
for u = 1 : col_count
    if rcc(u) > 0.00000001
    entering_index = u;                         % We are finding the first
index of RCC that is greater than zero.
    enter = rcc(u)
    break
    end
end
else
for t = 1 : col_count
    if rcc(t)== enter
    entering_index = t;                         % If not degenerate, normal
simplex is happening here
    break
    end
end
end

% End of this degeneracy code and simplex continues



%% Stage 9 : We check whether the LP is bounded or not.

ubc = zeros(col_count,1);                      % ubc is unbounded check

ubc(iB) = inv(mB)*A(:,entering_index);

var = max(ubc);                                % We assign max value of ubc
to var

if var < 0.000000001                           % if var < 0, it's
unbounded;
fprintf('LP is unbounded');
break
end


for e= 1:col_count
```

```
        if ubc(e) <= 0.0000001                     % If the Lp is bounded we
need to find the ratio for all POSITIVE values of ubc
                                                   % and to choose the minimum
                                                   % of that.. So assign all
                                                   % NON POSITIVE values to
                                                   % INFINITY in ubc
        ubc(e) = Inf;
        end
end
                                                    % here we the ratio and
                                                    % also the minimum of that.

for d = 1:col_count
if ubc(d) > 0.000000001 && ubc(d) ~= inf
ubc(d) = x(d)/ubc(d);
end
end

leave= min(ubc);

%% Stage 10: If the problem is degenerate, the code uses Bland's rule to
select a variable to leavethe basis as per the following:
if count > 1
for r = 1 : col_count
    if ubc(r)== leave
                                            % if there is degeneracy leave = 0
    exiting_index= r;                       % We are finding the first index of
ubc whose rato is zero is greater than zero.
    break
    end
end
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);
    iB(t) = entering_index;                 % Here the index of entering column
replaces the index of leaving column in Basis
    break
    end
end
else
[leave exiting_index] = min(ubc);
for t=1:row_count
    if iB(t) == exiting_index
    k = iB(t);                              % Else normal simplex is happening
here and index of entering column
                                            % replaces the index of leaving
column in Basis
    iB(t) = entering_index;
    break
    end
end
end

% End of this degeneracy code and simplex continues


iB;
mB = A(:, iB);

end
```

```
end                                         % for the if just above second
while

end                                         % for the if to move from phase 1
to phase 2
```

OUTPUTS:

```
current basis is optimal
x =

   20.0000
   30.0000
   20.0000
         0
   20.0000
   10.0000
   20.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.0000
         0
    0.0000
```

```
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
   40.0000
   40.0000
   40.0000
   40.0000
   40.0000
   40.0000
   40.0000
   40.0000
```

```
Zopt =

   910
```

Results given by COMMERCIAL SOLVER:

Inputs:

```
Trial>> Aeq = [-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;
0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;]
```

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

Beq = [20 30 50 60 20 50 30 60 40 0 0 0 40 40 0 0 0 40]

b = [0 0 0 0 0 0 0 0]

c = [8 8 8 8 7 7 7 7 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 110 160 110 160 110 160 110 160 135 195 135 195 135 195 135 195 0 0 0 0 0

lb = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
ub = []
[x z] = linprog(c,A,b,Aeq,Beq,lb,ub)
```

Result:

```
x =

    20
    30
    20
     0
    20
    10
    20
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
```

```
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
    40
    40
    40
    40
    40
    40
    40
    40

z =

   910
```

Hence the optimal objective value obtained through the code is the same with that of Commercial solver's. Hence the code is verified.