FALL SEM – (2020-21)

MAT2003

SUBMITTED BY : SRIHARSHITHA DEEPALA

REG NO : 19BCD7246

Question -1 :

Mohan-Meakins Breveries Ltd. has two bottling plants, one located at Solan and the other at Mohan Nagar. Each plant produces three drinks, whisky, beer and fruit juices named A, B and C respectively. The number of bottles produced per day are as follows:

	Plant at Solan (S)	Plant at Mohan Nagar (M)
Whisky, A	1,500	1,500
Beer, B	3,000	1,000
Fruit juices, C	2,000	5,000

A market survey indicates that during the month of April, there will be a demand of 20,000 bottles of whisky, 40,000 bottles of beer and 44,000 bottles of fruit juices. The operating costs per day for plants at Solan and Mohan Nagar are 600 and 400 monetary units. For how many days each plant be run in April so as to minimize the production cost, while still meeting the market demand ? [P.U.B.E. (Elect.) 2001; Pune U.MBA, 2000]

CODE:

(1) Finding optimal solution using linprog

```
clc
clear all
x = optimvar('x');
y = optimvar('y');
prob = optimproblem('Objective',600*x+400*y,'ObjectiveSense','min');
prob.Constraints.c1 = 1500*x + 1500*y>= 20000;
prob.Constraints.c2 = 3000*x + 1000*y>=40000;
prob.Constraints.c3 = 2000*x + 5000*y>=44000;
problem = prob2struct(prob);
[sol,fval] = linprog(problem);
disp("value of x is "+sol(1));
disp("value of y is "+sol(2));
fprintf("Min Z is %f ",fval);
```

OUTPUT:

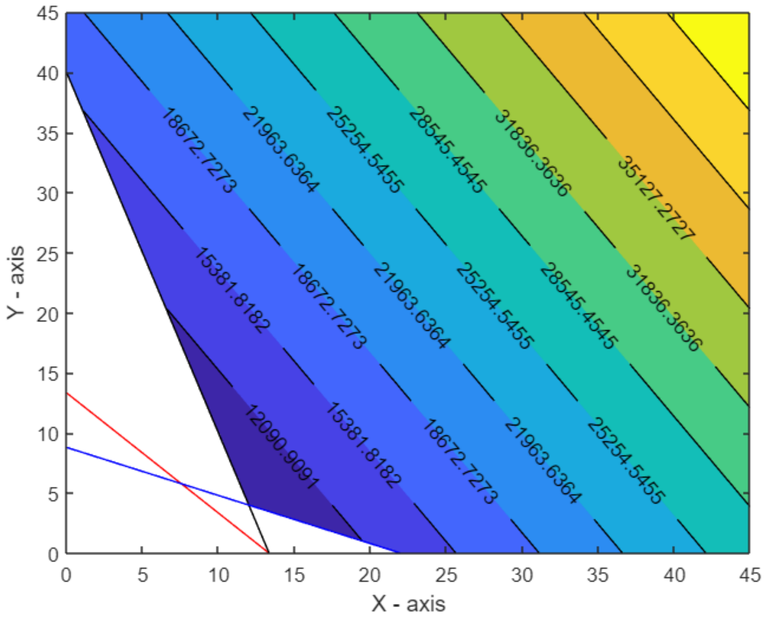
```
Optimal solution found.

value of x is 12
value of y is 4
Min Z is 8800.000000
>> |
```

(2) Shading Feasible solution

```
clear all
clc
%Finding Optimal Solution
%Generate data
[x,y] = meshgrid(0:0.1:45);
z = 600*x + 400*y;
cond1=1500*x + 1500*y >= 20000;
cond2=3000*x + 1000*y >= 40000;
cond3=2000*x + 5000*y >= 44000;
cond4=x>=0;
cond5=y>=0;
% Get boundaries of the condition
Cp1=(20000-1500*x(1,:))/1500;
Cp2=(40000-3000*x(1,:))/1000;
Cp3=(44000-2000*x(1,:))/5000;
%Remove the area where conditions doesn't apply
z(~cond1)=NaN;
z(~cond2)=NaN;
z(~cond3)=NaN;
[C,h]=contourf(x,y,z,10);
clabel(C,h,'LabelSpacing',100)
hold on
plot(x(1,:),Cp1,'r')
plot(x(1,:),Cp2,'k')
plot(x(1,:),Cp3,'b')
axis([0 45 0 45])
xlabel('X - axis')
ylabel('Y - axis')
```

OUTPUT:



Question – 2:

Use simplex method to solve the following problem :

$$\text{Maximize } Z = 2x_1 + 5x_2,$$

$$\begin{aligned} \text{subject to} \quad x_1 + 4x_2 &\le 24, \\ 3x_1 + x_2 &\le 21, \\ x_1 + x_2 &\le 9, \\ x_1, x_2 &\ge 0. \end{aligned}$$

CODE:

```
clc
clear all
No_of_Variables=2;
```

```
MaxZ= input('Max z :');
A=input('Matrix A');
B=input('Matrix B');
s=eye(size(A,1));
X=[A s B];
cost=input('Cost Matrix');
BV=No_of_Variables+1:1:size(X,2)-1;
zjCj=cost(BV)*X-cost;
zcj=[zjCj;X];
SimTable=array2table(zcj);
SimTable.Properties.VariableNames(1:size(zcj,2))={'x1','x2','s1','s2
','s3','Xb'}
repeat_iteration=true;
while repeat_iteration
if any(zjCj<0)
fprintf('Current solution is not optimal \n')
fprintf('Next Iteration: \n')
disp('Previous basic variables: ')
disp(BV)
zEntVar=zjCj(1:end-1);
[entrnCol,pivCol]=min(zEntVar);
fprintf("most min in zjcj is %d \n",entrnCol)
fprintf('The Pivot column is %d \n',pivCol)
Xb=X(:,end);
Col=X(:,pivCol);
if all(Col<0)
error('Lpp is unbounded as all entries less than zero')
 else
 for i=1:size(Col,1)
 if Col(i)>0
 minratio(i)=Xb(i)./Col(i);
 else
 minratio(i)=inf;
 end
 end
 [minrat,pvtRow]=min(minratio);
 fprintf('Minimum ratio corresponding to pivot row is %d \n',pvtRow)
 fprintf('Leaving variable is %d \n',BV(pvtRow))
 end
 BV(pvtRow)=pivCol;
 disp('New Basic Variables:')
 disp(BV)
 pvtKey=X(pvtRow,pivCol);
 X(pvtRow,:)=X(pvtRow,:)./pvtKey;
 for i=1:size(X,1)
 if i~=pvtRow
 X(i,:)=X(i,:)-X(i,pivCol).*X(pvtRow,:);
 end
 end
```

```
 zjCj=zjCj-zjCj(pivCol).*X(pvtRow,:);
 zcj=[zjCj;X];
 SimTable=array2table(zcj);

SimTable.Properties.VariableNames(1:size(zcj,2))={'x1','x2','s1','s2
','s3','Xb'}
 BFS=zeros(1,size(X,2));
 BFS(BV)=X(:,end);
 BFS(end)=sum(BFS.*cost);
 CurrentBFS=array2table(BFS);

CurrentBFS.Properties.VariableNames(1:size(CurrentBFS,2))={'x1','x2'
,'s1','s2','s3','Xb'}
 else
 repeat_iteration=false;
 fprintf('Current Basic feasible sol is optimal \n');
 end
end
```

OUTPUT:

```
Max z :
[2 5]
Matrix A
[1 4;3 1;1 1]
Matrix B
[24;21;9]
Cost Matrix
[2 5 0 0 0 0]

SimTable =

  4×6 table
```

x1	x2	s1	s2	s3	Xb
-2	-5	0	0	0	0
1	4	1	0	0	24
3	1	0	1	0	21
1	1	0	0	1	9

```
Current solution is not optimal
Next Iteration:
Previous basic variables:
     3     4     5


most min in zjcj is -5
The Pivot column is 2
Minimum ratio corresponding to pivot row is 1
Leaving variable is 3
New Basic Variables:
     2     4     5



SimTable =

  4×6 table
```

x1	x2	s1	s2	s3	Xb
-0.75	0	1.25	0	0	30
0.25	1	0.25	0	0	6
2.75	0	-0.25	1	0	15
0.75	0	-0.25	0	1	3

```
CurrentBFS =

  1×6 table
```

x1	x2	s1	s2	s3	Xb
0	6	0	15	3	30

```
Current solution is not optimal
```

```
Next Iteration:
Previous basic variables:
     2     4     5


most min in zjcj is -7.500000e-01
The Pivot column is 1
Minimum ratio corresponding to pivot row is 3
Leaving variable is 5
New Basic Variables:
     2     4     1
```

SimTable =

4×6 table

x1	x2	s1	s2	s3	Xb
0	0	1	0	1	33
0	1	0.33333	0	-0.33333	5
0	0	0.66667	1	-3.6667	4
1	0	-0.33333	0	1.3333	4

CurrentBFS =

1×6 table

x1	x2	s1	s2	s3	Xb
4	5	0	4	0	33

Current Basic feasible sol is optimal

. An Air Force is experimenting with three types of bombs P, Q and R in which three kinds of explosives, viz. A, B and C will be used. Taking the various factors into account, it has been decided to use at the maximum 600 kg of explosive A, at least 480 kg of explosive B and exactly 540 kg of explosive C. Bomb P requires 3, 2, 2 kg, bomb Q requires 1, 4, 3 kg and bomb R requires 4, 2, 3 kg of explosives A, B and C respectively. Bomb P is estimated to give the equivalent of a 2 ton explosion, bomb Q, a 3 ton explosion and bomb R, a 4 ton explosion respectively. Under what production schedule can the Air Force make the biggest bang ? VSC Big-M 6

CODE:

```
V=input('Variable names:')
M=1000;
C = input('Cost Matrix:');
A=input('Matrix A:')
s=eye(size(A,1));
BV=[];
for j=1:size(s,2)
   for i=1:size(A,2)
     if A(:,i)==s(:,j)
        BV=[BV i];
     end
   end
end
ZjCj=C(BV)*A-C;
ZCj=[ZjCj;A];
BigM= array2table(ZCj);
BigM.Properties.VariableNames(1:size(ZCj,2))=V
m=true;
while m
  ZC=ZjCj(:,1:end-1);
  if any(ZC<0)
    fprintf('The current Basic Feasible solution is not optimal\n');
    [Entval,pvt_col]=min(ZC);
    fprintf('Entering Column = %d\n',pvt_col);
    sol=A(:,end);
    Column=A(:,pvt_col);
    if all(Column<=0)
      fprintf('UNBOUNDED! ');
    else
     for i=1:size(Column,1)
       if Column(i)>0
         ratio(i)=sol(i)./Column(i);
```

```
        else
          ratio(i)=inf;
        end
      end
      [minR,pvt_row]=min(ratio);
      fprintf('Leaving Row = %d\n', pvt_row);
   end
   BV(pvt_row)=pvt_col;
   B=A(:,BV);
   A=inv(B)*A;
   ZjCj=C(BV)*A-C;
   ZCj=[ZjCj;A];
   BigM=array2table(ZCj);
   BigM.Properties.VariableNames(1:size(ZCj,2))=V
   else
     m=false;
     fprintf('The Optimal solution is reached ');
 Final_BFS = zeros(1,size(A,2));
 Final_BFS(BV) =A(:,end);
 Final_BFS(end)=sum(Final_BFS.*C);
 OptimalBFS=array2table(Final_BFS);
 OptimalBFS.Properties.VariableNames(1:size(OptimalBFS,2))=V
 end
 end
```

OUTPUT:

```
V=input('Variable names:')
M=1000;
C = input('Cost Matrix:');
A=input('Matrix A:')
s=eye(size(A,1));
BV=[];
for j=1:size(s,2)
   for i=1:size(A,2)
     if A(:,i)==s(:,j)
        BV=[BV i];
     end
   end
end
ZjCj=C(BV)*A-C;
ZCj=[ZjCj;A];
BigM= array2table(ZCj);
BigM.Properties.VariableNames(1:size(ZCj,2))=V
m=true;
while m
  ZC=ZjCj(:,1:end-1);
  if any(ZC<0)
    fprintf('The current Basic Feasible solution is not optimal\n');
    [Entval,pvt_col]=min(ZC);
```

```
    fprintf('Entering Column = %d\n',pvt_col);
    sol=A(:,end);
    Column=A(:,pvt_col);
    if all(Column<=0)
      fprintf('UNBOUNDED! ');
    else
     for i=1:size(Column,1)
       if Column(i)>0
         ratio(i)=sol(i)./Column(i);
       else
         ratio(i)=inf;
       end
     end
     [minR,pvt_row]=min(ratio);
     fprintf('Leaving Row = %d\n', pvt_row);
  end
  BV(pvt_row)=pvt_col;
  B=A(:,BV);
  A=inv(B)*A;
  ZjCj=C(BV)*A-C;
  ZCj=[ZjCj;A];
  BigM=array2table(ZCj);
  BigM.Properties.VariableNames(1:size(ZCj,2))=V
  else
    m=false;
    fprintf('The Optimal solution is reached ');
    Final_BFS = zeros(1,size(A,2));
    Final_BFS(BV) =A(:,end);
    Final_BFS(end)=sum(Final_BFS.*C);
    OptimalBFS=array2table(Final_BFS);
    OptimalBFS.Properties.VariableNames(1:size(OptimalBFS,2))=V
   end
end
```

```
Variable names:
{'x1', 'x2', 'x3', 's1', 's2', 'A1', 'A2', 'Sol'}


V =


  1×8 cell array


    {'x1'}    {'x2'}    {'x3'}    {'s1'}    {'s2'}    {'A1'}    {'A2'}
{'Sol'}


Cost Matrix:
[2 3 4 0 0 -M -M 0]
Matrix A:
```

```
[3 1 4 1 0 0 0 600;2 4 2 0 -1 1 0 480; 2 3 3 0 0 0 1 540]

A =

     3     1     4     1     0     0     0   600
     2     4     2     0    -1     1     0   480
     2     3     3     0     0     0     1   540


BigM =

  4×8 table
```

x1	x2	x3	s1	s2	A1	A2	Sol
-4002	-7003	-5004	0	1000	0	0	-1.02e+06
3	1	4	1	0	0	0	600
2	4	2	0	-1	1	0	480
2	3	3	0	0	0	1	540

```
The current Basic Feasible solution is not optimal
Entering Column = 2
Leaving Row = 2

BigM =

  4×8 table
```

x1	x2	x3	s1	s2	A1	A2	Sol
-500.5	0	-1502.5	0	-750.75	1750.8	0	-1.7964e+05
2.5	0	3.5	1	0.25	-0.25	0	480
0.5	1	0.5	0	-0.25	0.25	0	120
0.5	0	1.5	0	0.75	-0.75	1	180

```
The current Basic Feasible solution is not optimal
```

```
Entering Column = 3
Leaving Row = 3


BigM =


  4×8 table
```

x1	x2	x3	s1	s2	A1	A2	Sol
0.33333	0	0	0	0.5	999.5	1001.7	660
1.3333	0	4.4409e-16	1	-1.5	1.5	-2.3333	60
0.33333	1	2.7756e-17	0	-0.5	0.5	-0.33333	60
0.33333	0	1	0	0.5	-0.5	0.66667	120

```
The Optimal solution is reached
OptimalBFS =


  1×8 table
```

x1	x2	x3	s1	s2	A1	A2	Sol
0	60	120	60	0	0	0	660

Question – 4:

A product is produced by four factories A, B, C and D. The unit production costs in them are ₹ 2, ₹ 3, Re. 1 and ₹ 5 respectively. Their production capacities are: factory A – 50 units, B – 70 units, C – 30 units and D – 50 units. These factories supply the product to four stores, demands of which are 25, 35, 105 and 20 units respectively. Unit transport cost in rupees from each factory to each store is given in the table below.

TABLE 3.112

Stores

Factories	1	2	3	4
A	2	4	6	11
B	10	8	7	5
C	13	3	9	12
D	4	6	8	3

Determine the extent of deliveries from each of the factories to each of the stores so that the total production and transportation cost is minimum.

CODE:

```
% Vogel'Approximation Method (VAM) and Modified distribution method (MODI)
clc
clear all
c=input("Enter the cost Matrix");
[m,n]=size(c);
s=input("Enter the supplies for column vector");
d=input("Enter the demands for row vector");
r=0.1;% value of extra element in the corrective degeneracy problem
numbasic=m+n-1;
c1=zeros(m,n);
s1=zeros(m,1);
d1=zeros(1,n);
x=zeros(m,n);
% sum demand and supply
sumd=0;
for j=1:n
sumd=sumd+d(j);
end
sums=0;
for i=1:m
sums=sums+s(i);
end
% Checking balance of demand and supply
if sumd==sums
    disp('The problem is balanced');
else
disp('Review amount of supply and demand');
end
% Equivalant matrix
for j=1:n
for i=1:m
c1(i,j)=c(i,j);
end
end
for i=1:m
s1(i)=s(i);
end
for j=1:n
d1(j)=d(j);
end
toc
tic
%% Vogel's approximation method
iteration=0;
for k=1:m+n-1
iteration=iteration+1;
%% Row Difference
minrow1=zeros(m,1);
minrow2=zeros(m,1);
jmin=zeros(1,m);
for i=1:m
min1=inf;
for j=1:n
if c1(i,j)<min1
```

```
min1=c1(i,j);
jmin(i)=j;
end
end
minrow1(i)=min1;
end
for i=1:m
min2=inf;
for j=1:n
if j~=jmin(i)
if c1(i,j)<=min2
min2=c1(i,j);
end
end
end
minrow2(i)=min2;
end
%% Column Difference
mincol1=zeros(1,n);
mincol2=zeros(1,n);
imin=zeros(n,1);
for j=1:n
minR1=inf;
for i=1:m
if c1(i,j)<minR1
minR1=c1(i,j);
imin(j)=i;% position of minR1 each column
end
end
mincol1(j)=minR1;
end
for j=1:n
minR2=inf;
for i=1:m
if i~=imin(j)
if c1(i,j)<=minR2
minR2=c1(i,j);
end
end
end
mincol2(j)=minR2;
end
%% Difference
diffrow=zeros(m,1);
diffcol=zeros(1,n);
for i=1:m
diffrow(i)=minrow2(i)-minrow1(i);
end
for j=1:n
diffcol(j)=mincol2(j)-mincol1(j);
end
%% The greatest difference
R=0;
Row=zeros(m,1);
for i=1:m
if diffrow(i)>=R
```

```
R=diffrow(i);
iminrow=i; % the greatest diff. on column
end
end
Row(iminrow)=R;
S=0;
Col=zeros(1,n);
for j=1:n
if diffcol(j)>=S
S=diffcol(j);
jmincol=j;% the greatest diff. on row
end
end
Col(jmincol)=S;
great=zeros(1,n);
for j=1:n
if S>=R
great(jmincol)=Col(jmincol);
Colline=1;
else
great(iminrow)=Row(iminrow);
Colline=0;
end
end
%% Search the entry cell
if Colline==1 %Colline = 1
j=jmincol;
R1=inf;
for i=1:m
if c1(i,jmincol)<=R1
R1=c1(i,jmincol);
igreat=i; % the lowest cost on the jmincol
end
end
if s1(igreat)>d1(jmincol)
x(igreat,jmincol)=d1(jmincol);
s1(igreat)=s1(igreat)-d1(jmincol);
d1(jmincol)=0;
eliminaterow=0; % If current demand =0 (eliminaterow=0), eliminate a column.
elseif s1(igreat)<d1(jmincol)
x(igreat,jmincol)=s1(igreat);
d1(jmincol)=d1(jmincol)-s1(igreat);
s1(igreat)=0;
eliminaterow=1; % If supply =0 (eliminaterow=1), eliminate a row.
elseif s1(igreat)==d1(jmincol)
x(igreat,jmincol)=s1(igreat);
d1(jmincol)=0;
s1(igreat)=0;
eliminaterow=2;% If supply=demnad (eliminaterow=2),eliminate both a row and a column
end
% Eliminate a column or a row
if eliminaterow==0% Eliminate a column
for i=1:m
c1(i,jmincol)=inf;
end
elseif eliminaterow==1 % Eliminate a row
```

```
for j=1:n
c1(igreat,j)=inf;
end
elseif eliminaterow==2
for i=1:m
c1(i,jmincol)=inf;
end
for j=1:n
c1(igreat,j)=inf;
end
end
else % Colline=0;
i=iminrow;
R2=inf;
for j=1:n
if c1(iminrow,j)<R2
R2=c1(iminrow,j);
jgreat=j; % the lowest cost on the iminrow
end
end


if s1(iminrow)>d1(jgreat)
x(iminrow,jgreat)=d1(jgreat);
s1(iminrow)=s1(iminrow)-d1(jgreat);
d1(jgreat)=0;
eliminaterow=0; % If current demand=0 (eliminaterow=0), eliminate a column.
elseif s1(iminrow)<d1(jgreat)
x(iminrow,jgreat)=s1(iminrow);


d1(jgreat)=d1(jgreat)-s1(iminrow);
s1(iminrow)=0;
eliminaterow=1; % If current supply =0 (eliminaterow=1),eliminate a row
elseif s1(iminrow)==d1(jgreat)
x(iminrow,jgreat)=s1(iminrow);
d1(jgreat)=0;
s1(iminrow)=0;
eliminaterow=2; % If current supply =demand (eliminaterow=2),eliminate both a row and a
column
end
% Eliminate a column or a row
if eliminaterow==0% Eliminate a column
for i=1:m
c1(i,jgreat)=inf;% jmincol
end
elseif eliminaterow==1 % Eliminate a row
for j=1:n
c1(iminrow,j)=inf; % iminrow = the greatest diff. row
end
elseif eliminaterow==2 %Eliminate both a row and a column
for i=1:m
c1(i,jgreat)=inf;% Eliminate a column
end
for j=1:n
c1(iminrow,j)=inf; % Eliminate a row
```

```
end
end
end
%% Calculate the objective function
ZVAM=0;
for j=1:n
for i=1:m
if x(i,j)>0
ZVAM=ZVAM+c(i,j)*x(i,j);
end
end
end
end
%% The degeneracy
countx=0;
for i=1:m
for j=1:n
if x(i,j)>0
countx=countx+1;
x1(i,j)=x(i,j);
x2(i,j)=x(i,j);
end
end
end
if countx>=numbasic
degen=0;
disp('Total cost of Non-degeneracy VAM');
disp(ZVAM);
disp('Occupied matrix of VAM')
disp(x)
else
degen=1;
disp('Total cost of Degeneracy VAM');
disp(ZVAM);
disp('Occupied matrix of VAM')
disp(x)
end
toc
%% How to correct degeneracy matrix
if degen==1
numdegen=numbasic-countx;
iterationDegen=0;
for A=1:numdegen
iterationDegen=iterationDegen+1;
%% Construct the u-v variables
udual=zeros(m,1);
vdual=zeros(1,n);
udual(1)=0;
for i=1:1
for j=1:n
if x(i,j)>0
vdual(j)=c(i,j)-udual(i);
end
end
end
for j=1:1
```

```
for i=1:m
if x(i,j)>0
iu=i
udual(iu)=c(i,j)-vdual(j);
end
end
end
for k=1:m*n
for i=1:m
if udual(i)>0
iu=i;
for j=1:n
if x(iu,j)>0
vdual(j)=c(iu,j)-udual(iu);
end
end
end
end
for j=1:n
if vdual(j)>0
jv=j;
for i=1:m
if x(i,jv)>0
udual(i)=c(i,jv)-vdual(jv);
end
end
end
end
countu=0;
countv=0;
for i=1:m
if udual(i)<inf
countu=countu+1;
end
end;
for j=1:n;
if vdual(j)<inf;
countv=countv+1;
end
end
if (countu==m) && (countv==n)
return
end
end
%% Find the non-basic cells
unx=zeros(m,n);
for j=1:n
for i=1:m
if x(i,j)==0
unx(i,j)=c(i,j)-udual(i)-vdual(j)
end
end
end
%% Search maximum positive of udual+vdual-c(i,j) to reach a new basic variable
maxunx=inf;
for j=1:n
```

```
for i=1:m
if unx(i,j)<=maxunx
maxunx=unx(i,j);
imax=i;
jmax=j;
end
end
end
%% Count the number of basic variable on each and row
for j=1:n
sumcol=0;
for i=1:m
if x(i,j)>0
sumcol=sumcol+1;
end
end
x(m+1,j)=sumcol;
end
for i=1:m
sumrow=0;
for j=1:n
if x(i,j)>0
sumrow=sumrow+1;
end
end
x(i,n+1)=sumrow;
end
%% Construct the equipvalent x matrix
for j=1:n+1
for i=1:m+1
x1(i,j)=x(i,j);
end
end
%% Eliminate an entering variable for adding a new one
for j=1:n
for i=1:m
if (x(i,j)==1 || x1(i,j)==1)
x1(i,j)=0;
end
end
end
% Add a new entering variable to x1 matrix
% Assign the small value =1
for j=1:n;
for i=1:m
x1(imax,jmax)=1;
end
end
% Seaching and adding the entering point for corrective action
for i=1:m
if i~=imax
if x1(i,jmax)>0
ienter=i;
for j=1:n
if j~=jmax
if x1(ienter,j)>0 && x1(imax,j)==0
```

```
jenter=j;
end
end
end
end
end
end
x1(imax,jenter)=1;
x(imax,jenter)=1;
end
else
end
tic;
iterationOpt=0;
for q=1:n*m
iterationOpt=iterationOpt+1;
% The Modified distribution method
%% Construct the u-v variables
udual=zeros(m,1);
vdual=zeros(1,n);
for i=1:m
udual(i)=inf;
end
for j=1:n
vdual(j)=inf;
end
udual(1)=0;
for i=1:1
for j=1:n
if x(i,j)>0 && udual(i)<inf
vdual(j)=c(i,j)-udual(i);
end
end
end
for j=1:1
for i=1:m
if x(i,j)>0
iu=i;
if udual(iu)~=inf
vdual(j)=c(i,j)-udual(iu);
else
if vdual(j)~=inf
udual(iu)=c(i,j)-vdual(j);
end
end
end
end
end
for k=1:m*n
for i=1:m
if udual(i)~=inf
iu=i;
for j=1:n
if x(iu,j)>0 && udual(iu)<inf
vdual(j)=c(iu,j)-udual(iu);
end
```

```
end
end
end
for j=1:n
if vdual(j)~=inf
jv=j;
for i=1:m
if x(i,jv)>0 && vdual(jv)<inf
udual(i)=c(i,jv)-vdual(jv);
end
end
end
end
countu=0;
countv=0;
for i=1:m
if udual(i)<inf
countu=countu+1;
end
end
for j=1:n
if vdual(j)<inf
countv=countv+1;
end
end
if (countu==m) && (countv==n)
break
end
end
%% Find the non-basic cells
unx=zeros(m,n);
for j=1:n
for i=1:m
if x(i,j)==0
unx(i,j)=udual(i)+vdual(j)-c(i,j);
end
end
end
%% Search maximum positive of udual+vdual-c(i,j) to reach a new basic variable
maxunx=0;
for j=1:n
for i=1:m
if unx(i,j)>=maxunx
maxunx=unx(i,j);
imax=i;
jmax=j;
end
end
end
%% The objective function value
Z=0;
for j=1:n
for i=1:m
if x(i,j)>0
Z=Z+x(i,j)*c(i,j);
end
```

```
end
end
iterationOpt=iterationOpt+1;
%% Control loop
if maxunx==0
break;
else
end
%% Entering a new basic variable add into the basic variable matrix
x1=zeros(m+1,n+1);
x2=zeros(m+1,n+1);
% Construct the equivalent basic variable matrix
for j=1:n
for i=1:m
if x(i,j)>0
x1(i,j)=x(i,j);
x2(i,j)=x(i,j);
end
end
end
% Entering the new variable
x1(imax,jmax)=inf;
x2(imax,jmax)=inf;
for j=1:n
countcol=0;
for i=1:m
if x1(i,j)>0 || x1(i,j)==inf
countcol=countcol+1;
end
end
x1(m+1,j)=countcol;
x2(m+1,j)=countcol;
end
for i=1:m
countrow=0;
for j=1:n
if x1(i,j)>0
countrow=countrow+1;
end
end
x1(i,n+1)=countrow;
x2(i,n+1)=countrow;
end
%% Construct loop
% Eliminate the basic variables that has only one on each row
iterationloop=0;
for i=1:m
iterationloop=iterationloop+1;
for i=1:m
if x2(i,n+1)==1
ieliminate=i;
for j=1:n
if x2(ieliminate,j)<inf && x2(ieliminate,j)>0
jeliminate=j;
x2(ieliminate,jeliminate)=0;% Eliminate the basic variable on row
x2(ieliminate,n+1)=x2(ieliminate,n+1)-1; % decrease the number of the basic
```

```
variable on row one unit
x2(m+1,jeliminate)=x2(m+1,jeliminate)-1; % decrease the number of the basic
variable on column one unit
end
end
end
end
% Eliminate the basic variables that has only one on each column
for j=1:n
if x2(m+1,j)==1
jeliminate1=j;
for i=1:m
if x2(i,jeliminate1)<inf && x2(i,jeliminate1)>0
ieliminate1=i;
x2(ieliminate1,jeliminate1)=0;% Eliminate the basic variable on row
x2(m+1,jeliminate1)=x2(m+1,jeliminate1)-1; % decrease the number of the basic
variable on column one unit
x2(ieliminate1,n+1)=x2(ieliminate1,n+1)-1;% decrease the number of the basic
variable on row one unit
end
end
end
end
% Control the constructing loop path
for j=1:n
for i=1:m
if (x2(i,n+1)==0 || x2(i,n+1)==2) && (x2(m+1,j)==0 || x2(m+1,j)==2)
break;
else
end
end
end
end
%% Make +/-sign on basic variables in the loop path (x2)
%1. Add - sign on basic variable on row(imax) and on basic variable on
%column (jmax)
for j=1:n
if (x2(imax,j)~=0 && x2(imax,j)<inf && x2(imax,n+1)==2)
jneg=j;
x2(imax,jneg)=(-1)*x2(imax,jneg);
x2(m+1,jneg)=1;
x2(imax,n+1)=1;
for i=1:m
if (x2(i,jneg)>0 && x2(m+1,jneg)==1)
ineg=i;
end
end
end
end
for p=1:n
for j=1:n
if (j~=jneg && x2(ineg,j)>0 )&& (x2(ineg,n+1)==2)
jneg1=j;
x2(ineg,jneg1)=(-1)*x2(ineg,jneg1);
x2(ineg,n+1)=1;
x2(m+1,jneg1)=1;
```

```
for i=1:m
if (x2(i,jneg1)>0 && x2(m+1,jneg1)==1)
ineg1=i;
ineg=ineg1;
jneg=jneg1;
end
end
end
end
% Control loop
if jneg1==jmax
break
end
end
%% Search the net smallest negative basic variable in the loop path
small=inf;
for j=1:n
for i=1:m
if x2(i,j)<0
if abs(x2(i,j))<small
small=abs(x2(i,j));
end
end
end
end
% Construct the loop path
x3=zeros(m,n);
for j=1:n
for i=1:m
x3(i,j)=x2(i,j);
end
end
%% Add the smallest value to the positive basis variable and subtract to the
negative basic variable
for i=1:m
for j=1:n
x3(imax,jmax)=small;
if x3(i,j)~=0
if x3(i,j)<0
x3(i,j)=(x3(i,j))+small;
if x3(i,j)==0
x3(i,j)=inf;
end
else
if i~=imax && j~=jmax
x3(i,j)=x3(i,j)+small;
end
end
end
end
end
%% Combine the new absolute loop path to the x matrix
xpath=zeros(m,n);
for j=1:n
for i=1:m
xpath(i,j)=x(i,j);
```

```
end
end
for j=1:n
for i=1:m
if x3(i,j)~=0
if x3(i,j)==inf
xpath(i,j)=0;
else
xpath(i,j)=abs(x3(i,j));
end
end
end
end
%% The objective function
Zopt=0;
for j=1:n
for i=1:m
if round(xpath(i,j))>0
Zopt=Zopt+round(xpath(i,j))*c(i,j);
end
end
end
%% Check balance
sumbal=0;
for j=1:n
for i=1:m
if xpath(i,j)>0
sumbal=sumbal+xpath(i,j);
end
end
end
if sums==sumbal
disp('Balance');
else
break;
end
%% Transfer x to xpath
for j=1:n
for i=1:m
x(i,j)=xpath(i,j);
end
end
%% Checking degeneracy
countxMODI1=0;
for j=1:n
for i=1:m
if x(i,j)>0
countxMODI1=countxMODI1+1;
end
end
end
if countxMODI1<numbasic
degen2=1;
disp('Degeneracy within Loop');
disp(q);
else
```

```
degen2=0;
end
%% Corrective Degeneracy
if degen2==1
numdegen2=numbasic-countxMODI1;
iterationDegen2=0;
for A=1:numdegen2
iterationDegen2=iterationDegen2+1;
%% Construct the u-v variables
udual2=zeros(m,1);
vdual2=zeros(1,n);
for i=1:m
udual2(i)=inf;
end
for j=1:n
vdual2(j)=inf;
end
udual2(1)=0;
for i=1:1
for j=1:n
if x(i,j)>0
vdual2(j)=c(i,j)-udual2(i);
end
end
end
for j=1:1
for i=1:m
if x(i,j)>0
iu=i;
if udual2(iu)~=inf
vdual2(j)=c(i,j)-udual2(iu);
else
if vdual2(j)~=inf
udual2(iu)=c(i,j)-vdual2(j);
end
end
end
end
end
for k=1:m*n
for i=1:m
if udual2(i)~=inf
iu=i;
for j=1:n
if x(iu,j)>0
vdual2(j)=c(iu,j)-udual2(iu);
end
end
end
end
for j=1:n
if vdual2(j)~=inf
jv=j;
for i=1:m
if x(i,jv)>0
udual2(i)=c(i,jv)-vdual2(jv);
```

```
end
end
end
end
countu2=0;
countv2=0;
for i=1:m
if udual2(i)<inf
countu2=countu2+1;
end
end
for j=1:n
if vdual2(j)<inf
countv2=countv2+1;
end
end
if (countu2==m) & (countv2==n)
break
end
end
%% Find the non-basic cells
unx2=zeros(m,n);
for j=1:n
for i=1:m
if x(i,j)==0
unx2(i,j)=udual2(i)+vdual2(j)-c(i,j);
end
end
end
%% Search maximum positive of udual+vdual-c(i,j) to reach a new basic variable
maxunx2=0;
for j=1:n
for i=1:m
if unx2(i,j)>=maxunx2
maxunx2=unx2(i,j);
imax2=i;
jmax2=j;
end
end
end
%% Count the number of basic variable on each and row
for j=1:n
sumcol=0;
for i=1:m
if x(i,j)>0
sumcol=sumcol+1;
end
end
x(m+1,j)=sumcol;
end
for i=1:m
sumrow=0;
for j=1:n
if x(i,j)>0
sumrow=sumrow+1;
end
```

```
end
x(i,n+1)=sumrow;
end
%% Construct the equivalent x matrix
x12=zeros(m+1,n+1);
for j=1:n+1
for i=1:m+1
x12(i,j)=x(i,j);
end
end
%% Eliminate an entering variable for adding a new one
for j=1:n
for i=1:m
if (x(i,j)==r|| x12(i,j)==r)
x12(i,j)=0;
end
end
end
% Add a new entering variable to x1 matrix
for j=1:n
for i=1:m
x12(imax2,jmax2)=r;
end
end
% Seaching and adding the entering point for corrective action
for i=1:m
if i~=imax2
if x1(i,jmax2)>0
ienter2=i;
for j=1:n
if j~=jmax2
if x1(ienter2,j)>0 && x1(imax2,j)==0
jenter2=j;
end
end
end
end
end
end
x1(imax2,jenter2)=r;
x(imax2,jenter2)=r;
end
end% End of degen2==1;
end% End of q=1:n;
%% Checking degeneracy
countxMODI2=0;
for j=1:n
for i=1:m
if x(i,j)>0
countxMODI2=countxMODI2+1;
end
end
end
%% Total cost
ZMODI=0;
for j=1:n
```

```
for i=1:m
if x(i,j)>0
ZMODI=ZMODI+x(i,j)*c(i,j);
end
end
end
if countxMODI2<numbasic
disp('Degeneracy in solution of MODI');
end
if maxunx==0
disp('Total cost of MODI');
disp(ZMODI);
disp('Occupied matrix of MODI');
disp(x);
else
disp('Reviewed Loop');
end
toc
```

OUTPUT:

```
Enter the cost Matrix
[2+2,4+2, 6+2,11+2, 0;10+3,8+3,7+3,5+3, 0;13+1, 3+1,9+1,12+1, 0;4+5, 6+5, 8+5,
3+5, 0]
Enter the supplies for column vector
[50;70;30;50]
Enter the demands for row vector
[25 35 105 20 15]
The problem is balanced
Elapsed time is 294.428465 seconds.
Total cost of Non-degeneracy VAM
        1465

Occupied matrix of VAM
    25     5    20     0     0
     0     0    70     0     0
     0    30     0     0     0
     0     0    15    20    15

Elapsed time is 0.022143 seconds.
Total cost of MODI
        1465
```

```
Occupied matrix of MODI
    25     5    20     0     0
     0     0    70     0     0
     0    30     0     0     0
     0     0    15    20    15

Elapsed time is 0.011738 seconds.
```

Question – 5:

Five wagons are available at stations 1, 2, 3, 4 and 5. These are required at five stations I, II, III, IV and V. The mileages between various stations are given by the table below. How should the wagons be transported so as to minimize the total mileage covered?

TABLE 4.28

	I	II	III	IV	V
1	10	5	9	18	11
2	13	9	6	12	14
3	3	2	4	4	5
4	18	9	12	17	15
5	11	6	14	19	10

CODE:

```
function [assignment,cost] = Assign_prob(costMat)
costMat = input("enter the cost matrix")


assignment = zeros(1,size(costMat,1));
cost = 0;
validMat = costMat == costMat & costMat < Inf;
bigM = 10^(ceil(log10(sum(costMat(validMat))))+1);
costMat(~validMat) = bigM;
validCol = any(validMat,1);
validRow = any(validMat,2);


nRows = sum(validRow);
nCols = sum(validCol);
n = max(nRows,nCols);
if ~n
    return
end


maxv=10*max(costMat(validMat));
```

```
dMat = zeros(n) + maxv;
dMat(1:nRows,1:nCols) = costMat(validRow,validCol);
minR = min(dMat,[],2);
minC = min(bsxfun(@minus, dMat, minR));
zP = dMat == bsxfun(@plus, minC, minR);


starZ = zeros(n,1);
while any(zP(:))
    [r,c]=find(zP,1);
    starZ(r)=c;
    zP(r,:)=false;
    zP(:,c)=false;
end


while 1
    if all(starZ>0)
        break
    end
    coverColumn = false(1,n);
    coverColumn(starZ(starZ>0))=true;
    coverRow = false(n,1);
    primeZ = zeros(n,1);
    [rIdx, cIdx] =
find(dMat(~coverRow,~coverColumn)==bsxfun(@plus,minR(~cove
rRow),minC(~coverColumn)));
    while 1

cR = find(~coverRow);
cC = find(~coverColumn);
rIdx = cR(rIdx);
cIdx = cC(cIdx);
Step = 6;
while ~isempty(cIdx)
uZr = rIdx(1);
uZc = cIdx(1);
primeZ(uZr) = uZc;
stz = starZ(uZr);
if ~stz
Step = 5;
```

```
break;
end
coverRow(uZr) = true;
coverColumn(stz) = false;
z = rIdx==uZr;
rIdx(z) = [];
cIdx(z) = [];
cR = find(~coverRow);
z = dMat(~coverRow,stz) == minR(~coverRow) + minC(stz);
rIdx = [rIdx(:);cR(z)];
cIdx = [cIdx(:);stz(ones(sum(z),1))];
end
if Step == 6
[minval,rIdx,cIdx]=outerplus(dMat(~coverRow,~coverColumn),
minR(~coverRow),minC(~coverColumn));
minC(~coverColumn) = minC(~coverColumn) + minval;
minR(coverRow) = minR(coverRow) - minval;
else
break
end
end
rowZ1 = find(starZ==uZc);
starZ(uZr)=uZc;
while rowZ1>0
starZ(rowZ1)=0;
uZc = primeZ(rowZ1);
uZr = rowZ1;
rowZ1 = find(starZ==uZc);
starZ(uZr)=uZc;
end
end


rowIdx = find(validRow);
colIdx = find(validCol);
starZ = starZ(1:nRows);
vIdx = starZ <= nCols;
assignment(rowIdx(vIdx)) = colIdx(starZ(vIdx));
pass = assignment(assignment>0);
pass(~diag(validMat(assignment>0,pass))) = 0;
assignment(assignment>0) = pass;
```

```
cost =
trace(costMat(assignment>0,assignment(assignment>0)));
disp('Displaying the assignment:')
disp(assignment)
disp('Displaying total cost:')
disp(cost)
function [minval,rIdx,cIdx]=outerplus(M,x,y)
ny=size(M,2);
minval=inf;
for c=1:ny
M(:,c)=M(:,c)-(x+y(c));
minval = min(minval,min(M(:,c)));
end
[rIdx,cIdx]=find(M==minval);
```

OUTPUT:

```
costMat =


    10     5     9    18    11
    13     9     6    12    14
     3     2     4     4     5
    18     9    12    17    15
    11     6    14    19    10

Displaying the assignment:
     1     3     4     2     5

Displaying total cost:
    39



ans =


     1     3     4     2     5
```

Question – 6:

Solve by dual simplex method the following problem :

$$\text{Minimize} \quad Z = 2x_1 + 2x_2 + 4x_3,$$
$$\text{subject to} \quad 2x_1 + 3x_2 + 5x_3 \geq 2,$$
$$3x_1 + x_2 + 7x_3 \leq 3$$
$$x_1 + 4x_2 + 6x_3 \leq 5,$$
$$x_1, x_2, x_3 \geq 0.$$

CODE:

```
format short
clear all
clc
coltxt = input('Enter the array of variables');
cost = input('Enter the cost matrix :');
info = input('Enter the matrix of basic and nonbasic variables :');
b = input('Enter the solution matrix :');
s = eye(size(info,1));
A = [info s b];
BV = [];
for j = 1:size(s,2)
    for i = 1:size(A,2)
        if A(:,i)==s(:,j)
            BV = [BV i];
        end
    end
end
fprintf('Basic Variables (BV) = ');
disp(coltxt(BV));
ZjCj = cost(BV)*A - cost;
ZCj = [ZjCj;A];
simpTable = array2table(ZCj);
simpTable.Properties.VariableNames(1:size(ZCj,2))=coltxt;
disp(simpTable);
RUN =true;
while RUN
SOL = A(:,end);
if any(SOL<0)
fprintf("The current solution is not feasible\n");
[LeaVal,pvt_row]=min(SOL);
```

```
fprintf("Leaving row = %d\n",pvt_row);
ROW = A(pvt_row,1:end-1);
ZJ =ZjCj(:,1:end-1);
for i = 1:size(ROW,2)
if ROW(i)<0
ratio(i)=abs(ZJ(i)./ROW(i));
else
ratio(i) = inf;
end
end
[minVAL,pvt_col] = min(ratio);
fprintf("Entering variable = %d\n",pvt_col);
BV(pvt_row) = pvt_col;
fprintf("Basic Variables (BV) = ");
disp(coltxt(BV));
pvt_key = A(pvt_row,pvt_col);
A(pvt_row,:)=A(pvt_row,:)./pvt_key;
for i = 1:size(A,1)
if i~=pvt_row
A(i,:) = A(i,:)-A(i,pvt_col).*A(pvt_row,:);
end
end
ZjCj = cost(BV)*A - cost;
ZCj = [ZjCj;A];
simpTable = array2table(ZCj);
simpTable.Properties.VariableNames(1:size(ZCj,2))=coltxt;
disp(simpTable)
else
RUN =false;
fprintf("The current solution is feasible and optimal\n");
end
end
Final_BFS = zeros(1,size(A,2));
Final_BFS(BV) =A(:,end);
Final_BFS(end)=sum(Final_BFS.*cost);
OptimalBFS=array2table(Final_BFS);
OptimalBFS.Properties.VariableNames(1:size(OptimalBFS,2))=coltxt;
disp(OptimalBFS);
```

OUTPUT:

```
Enter the array of variables
{'x1', 'x2', 'x3', 's1', 's2', 's3','Sol'}
Enter the cost matrix :
[2 2 4 0 0 0 0]
Enter the matrix of basic and nonbasic variables :
[-2, -3, -5; 3 1 7; 1 4 6]
Enter the solution matrix :
[-2;3;5]
Basic Variables (BV) =     {'s1'}    {'s2'}    {'s3'}

    x1     x2     x3     s1     s2     s3     Sol
    __     __     __     __     __     __     ___

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


The current solution is not feasible
Leaving row = 1
Entering variable = 2
Basic Variables (BV) =     {'x2'}    {'s2'}    {'s3'}


       x1         x2        x3           s1        s2     s3       Sol
    ________     __     ________     ________     __     __     _______


    -0.66667     0      -0.66667     -0.66667     0      0       1.3333
     0.66667     1        1.6667     -0.33333     0      0      0.66667
      2.3333     0        5.3333      0.33333     1      0       2.3333
     -1.6667     0      -0.66667       1.3333     0      1       2.3333


The current solution is feasible and optimal
    x1       x2        x3     s1       s2         s3         Sol
    __    _______     __     __     ______     ______     ______


    0     0.66667     0      0      2.3333     2.3333     1.3333
```

Question – 7:

$$\text{Minimize } Z = y_1^2 + y_2^2 + y_3^2,$$
$$\text{subject to} \quad y_1 + y_2 + y_3 \geq 15,$$
$$y_1, y_2, y_3 \geq 0.$$

CODE:

```
%% Function file named dynopt
function [c, ceq] = dynopt(y)
  c = 0;
  ceq = sum(y)-15;
end


%% Solving the objective function by calling dynopt function
warning('off','all')
objective = @(y)y(1)^2 + y(2)^2+y(3)^2 ;
y0 = [0 0 0];
lb = 0*ones(3);
ub = inf*ones(3);
disp(['Initial Objective: ' num2str(objective(y0))])
A = [];
b = [];
Aeq = [];
beq = [];
nonlincon = @dynopt;
x = fmincon(objective,y0,A,b,Aeq,beq,lb,ub,nonlincon);
disp(['Final Objective: ' num2str(objective(x))])
disp('Solution')
disp(['y1 = ' num2str(x(1))])
disp(['y2 = ' num2str(x(2))])
disp(['y3 = ' num2str(x(3))])
```

OUTPUT:

```
>> main
Initial Objective: 0

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
Final Objective: 75
Solution
y1 = 5
y2 = 5
y3 = 5
>>
```

Question – 8:

A and B play a game in which each has three coins a 5p, a 10p and a 20p. Each player selects a coin without the knowledge of the other's choice. If the sum of the coins is an odd amount, A wins B's coin; if the sum is even, B wins A's coin. Find the best strategy for each player and the value of the game. [Univ. of Mumbai PGDM, 2012; J.N.T.U. Hyerabad B.Tech. (C.Sc.) Dec., 2011;

CODE:

```
A=input('Enter the Game Matrix:');
r=[];s=[];[m,n]=size(A);
if min(max(A))==max(min(A'))
    b=max(A);Strategy_Ist=[];Strategy_IInd=[];ms=[];
    for i=1:n
        for j=1:m
            if isequal(b(i),A(j,i))
                if isequal(A(j,i),min(A(j,:)))
                    r(length(r)+1)=j;
                    s(length(s)+1)=i;
                end
            end
        end
    end
    if (length(r)==1 && length (s)==1)
        Answer=['The Game has a saddle point at the location :- ('
int2str(r) ',' int2str(s) ') and value of the game is '
num2str(A(r,s),6) '. So no mixed strategy is needed.'];
    else
        for i=1:length(r)
            ms=[ms '(' int2str(r(i)) ',' int2str(s(i)) '),'];
        end
        Answer=['The Game has saddle points at the locations :-' ms
' and value of the game is ' num2str(A(r(1),s(1)),6) '. So no mixed
strategy is needed.'];
    end
else
    X_a=linprog(-[1;zeros(m,1)],[ones(n,1) -A'],zeros(n,1),[0
ones(1,m)],[1],[-inf;zeros(m,1)]);v=X_a(1,1);X_a(1,:)=[];
    X_b=linprog([1;zeros(n,1)],[-ones(m,1) A],zeros(m,1),[0
ones(1,n)],[1],[-inf;zeros(n,1)]);X_b(1,:)=[];
    Answer=['The Game has no saddle point and value of the game is '
num2str(v,6) ' and therefore the suggested mixed strategy is given
in mixed strategy matrix.'];
    disp(Answer);
    Strategy_Ist=X_a;
    disp('Strategy of A:')
    disp(Strategy_Ist)
```

```
    Strategy_IInd=X_b;
    disp('Strategy of B:');
    disp(Strategy_IInd);
end
```

OUTPUT:

```
>> game
Enter the Game Matrix:
[-5 10 20; 5 -10 -10; 5 -20 -20]

Optimal solution found.


Optimal solution found.

The Game has no saddle point and value of the game is 0 and therefore the suggested mixed strategy is given in mixed strategy matrix.
Strategy of A:
    0.5000
    0.5000
         0

Strategy of B:
    0.6667
    0.3333
         0

>>
```