EXPERIMENT 1

AIM : Simplex technique to solve LPP and reading dual solution from the optimal table.

SOURCE CODE :

```
clc;
//-----------INPUT PARAMETERS
Noofvariables=3;
c=[-1 3 -2]
info = [3 -1 2; -2 4 0; -4 3 8];
b =[ 7; 12 ;10];
n= size(info,"r");
s = eye(n,n);
A =[info s b];
//disp(A);
cost= zeros(1,size(A,"c"));
cost(1:Noofvariables)= c;

//---------------Constrain BV
BV = Noofvariables+1:size(A,2)-1;

// ---------------Calculate ZjCj
ZjCj = cost(BV)*A - cost;
//disp(ZjCj);
ZCj =[ZjCj;A];
mprintf('\n =========== Simplex Table ============\n')
disp(['x1' 'x2' 'x3' 's1' 's2' 's3' "Sol"], [   ZCj(1:4,1) ,
ZCj(1:4,2),ZCj(1:4,3),ZCj(1:4,4),ZCj(1:4,5),ZCj(1:4,6),ZCj(1:4,
7)]);
//simplex table start
// check any negative value
for i= 1:size(ZjCj,"c")
if (ZjCj(i) <0) then
    mprintf('\n    The current BFS is NOT Optimal \n')
```

```
   mprintf('\n ==== The NEXT ITERATION
RESULTS==========\n')
   end
end
disp("Old Basic Variable =  ");
disp(BV);
// Finding the Entering Variable
ZC = ZjCj(1:size(ZjCj,"c")-1);
[EnterCol,pvt_col] = min(ZC);
      mprintf('\n The Minimum element in Zj-Cj is %d
Corresponding to Column %d \n',EnterCol,pvt_col);

//Finding the Leaving Variable
sol= A(:,$);
Column =A(:,pvt_col);
z = size(Column,"r");
 j=0
for i=1:size(Column,"r")
   if Column(i)<0
   j=j+1;
   end
end
// To check UNBOUNDED
disp(j);
if j == z;
   mprintf(' LPP is UNBOUNDED. All entries <= 0 in
column  %d \n',pvt_col);
end

for i=1:size(Column,"r")
    if Column(i)>0
     ratio(i) =sol(i)./Column(i);
   else
     ratio(i)=%inf;
    end
end
```

```
[MinRatio,pvt_row] = min(ratio);
mprintf('\n The Minimum Ratio Corresponding to
PIVOT Row  %d \n',pvt_row);
//n=pvt_row
disp([' LEAVING variable is'] ,[BV(pvt_row)]);
BV(pvt_row)=pvt_col;
disp(" New Basic Variable (BV)= ");
disp(BV);

//KEY ELEMENT
pvt_key = A(pvt_row,pvt_col);
//disp(pvt_key);

//UPDATE THE table for NEXT ITERATION
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

ZjCj = ZjCj-ZjCj(pvt_col).*A(pvt_row,:);
ZCj = [ZjCj;A];
mprintf('\n =========== Next Iteration
============\n')
disp(['x1' 'x2' 'x3' 's1' 's2' 's3' "Sol"], [   ZCj(1:4,1) ,
ZCj(1:4,2),ZCj(1:4,3),ZCj(1:4,4),ZCj(1:4,5),ZCj(1:4,6),
ZCj(1:4,7)]);
end
BFS = zeros(1,size(A,2));
BFS(BV) = A(:,$);
BFS($) = sum(BFS.*cost);
mprintf('\n  ============The BFS is
================= \n')
```

disp(['x1' 'x2' 'x3' 's1' 's2' 's3' "Sol"], [BFS(1,1) , BFS(1,2),BFS(1,3),BFS(1,4),BFS(1,5),BFS(1,6),BFS(1,7)]);

OUTPUT :

Scilab 6.1.1 Console

```
=========== Simplex Table ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

  1.  -3.   2.   0.   0.   0.   0.
  3.  -1.   2.   1.   0.   0.   7.
 -2.   4.   0.   0.   1.   0.   12.
 -4.   3.   8.   0.   0.   1.   10.

   The current BFS is NOT Optimal

==== The NEXT ITERATION RESULTS==========

 "Old Basic Variable =  "

  4.   5.   6.

The Minimum element in Zj-Cj is -3 Corresponding to Column 2

  1.

The Minimum Ratio Corresponding to PIVOT Row  2

 " LEAVING variable is"

  5.

 " New Basic Variable (BV)= "

  4.   2.   6.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

 -0.5   0.   2.   0.   0.75   0.   9.
  2.5   0.   2.   1.   0.25   0.   10.
 -0.5   1.   0.   0.   0.25   0.   3.
```

```
 -4.     3.    8.    0.    0.      1.    10.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

 -0.5   0.    2.    0.    0.75   0.    9.
  2.5   0.    2.    1.    0.25   0.    10.
 -0.5   1.    0.    0.    0.25   0.    3.
 -4.    3.    8.    0.    0.      1.    10.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

 -0.5   0.    2.    0.    0.75   0.    9.
  2.5   0.    2.    1.    0.25   0.    10.
 -0.5   1.    0.    0.    0.25   0.    3.
 -2.5   0.    8.    0.   -0.75   1.    1.

 ============The BFS is ================

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

  0.    3.    0.    10.    0.    1.    9.
```

EXPERIMENT 2

AIM : Dual Simplex technique to solve LPP.

SOURCE CODE :

```
clc;
Variables =['x1','x2','x3','s1','s2','Sol'];
cost=[-2 0 -1 0 0 0];
info = [-1 -1 1; -1 2 -4];
b =[-5;-8];
n= size(info,"r");
s = eye(n,n);
A =[info s b];
//----------Finding starting BFS
BV=[];
for j=1:size(s,2)
   for i=1:size(A,2)
      if A(:,i)==s(:,j)
       BV = [BV i];
      end
   end
end

mprintf('\n Basic Variables (BV) = \n');
disp([Variables(BV)]);
ZjCj =cost(BV)*A - cost;
mprintf('\nZjCj = ');
disp(ZjCj);

// for print table
ZCj =[ZjCj;A];

//mprintf('\n =========== DUAL Simplex Table
============\n')
```

```
disp(['x1' 'x2' 'x3' 's1' 's2' "Sol"], [   ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5),ZCj(1:3,6)]);

// DUAL SIMPLEX START
RUN = 1;
while RUN

 Sol = A(:,$);
for i= 1:size(Sol,2)
   if (Sol(i) <0) then
   mprintf('\n=== The current BFS is NOT FEASIBLE ===\n')
//Finding Leaving Variable
   [LeaVal,pvt_row] = min(Sol);
   mprintf('\nLeaving Row = %d \n',pvt_row);

//Finding Entering Variable
   Row =A(pvt_row,1:$-1);
   ZJ = ZjCj(:,1:$-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);
 mprintf('\nEntering Variable = %d \n',pvt_col);

//Updating the BV
BV(pvt_row) = pvt_col;
mprintf('\nBasic Variables (BV) = ')
disp([Variables(BV)]);
//Update the table for Next Iteration

pvt_key=A(pvt_row,pvt_col);
//disp(pvt_key);
```

```
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
ZjCj = cost(BV)*A-cost
mprintf('\nZjCj = ');
disp(ZjCj);

//for print table
ZCj = [ZjCj;A];
 mprintf('\n =========== Next Iteration ============\n')
disp(['x1' 'x2' 'x3' 's1' 's2' "Sol"], [   ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5),ZCj(1:3,6)]);
//disp(A);
end

else
   RUN =0;
   mprintf('\n=== The current BFS is FEASIBLE & OPTIMAL
===\n')
   end
end
end
```

OUTPUT :

Scilab 6.1.1 Console

```
=========== Simplex Table ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

  1.  -3.   2.   0.   0.   0.   0.
  3.  -1.   2.   1.   0.   0.   7.
 -2.   4.   0.   0.   1.   0.   12.
 -4.   3.   8.   0.   0.   1.   10.

   The current BFS is NOT Optimal

==== The NEXT ITERATION RESULTS==========

 "Old Basic Variable =  "

  4.   5.   6.

The Minimum element in Zj-Cj is -3 Corresponding to Column 2

  1.

The Minimum Ratio Corresponding to PIVOT Row  2

 " LEAVING variable is"

  5.

 " New Basic Variable (BV)= "

  4.   2.   6.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "s3"  "Sol"

 -0.5   0.   2.   0.   0.75   0.   9.
  2.5   0.   2.   1.   0.25   0.   10.
 -0.5   1.   0.   0.   0.25   0.   3.
```

```
cilab 6.1.1 Console

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "Sol"

  1.75   0.5   0.    0.    0.25  -2.
 -1.25  -0.5   0.    1.    0.25  -7.
  0.25  -0.5   1.    0.   -0.25   2.

== The current BFS is NOT FEASIBLE ===

eaving Row = 1

ntering Variable = 2

asic Variables (BV) =
 "x2"  "x3"

jCj =
  1.75   0.5   0.    0.    0.25  -2.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "Sol"

  1.75   0.5   0.    0.    0.25  -2.
  2.5    1.    0.   -2.   -0.5    14.
  0.25  -0.5   1.    0.   -0.25   2.

jCj =
  0.5   0.   0.   1.   0.5  -9.

=========== Next Iteration ============

 "x1"  "x2"  "x3"  "s1"  "s2"  "Sol"

  0.5   0.   0.   1.   0.5  -9.
  2.5   1.   0.  -2.  -0.5   14.
  1.5   0.   1.  -1.  -0.5   9.
```

EXPERIMENT 3

AIM : Illustration of following special cases in LPP using Simplex Method .

UNRESTRICTED VARIABLE :

SOLVE LPP Max z = x(0)1 +3x(2) St = x(0) +x(2) <= 2   Here, x(0) = x-x(1) as x(0) is unrestricted -x(0)+x(2) <=4 x(2) >=0

SOURCE CODE :

```
clc;
//-----------INPUT PARAMETERS
Noofvariables=3;
c=[1 -1 3]
info = [1 -1 1; -1 1 1];
b =[ 2; 4];
n= size(info,"r");
s = eye(n,n);
A =[info s b];
//disp(A);
cost= zeros(1,size(A,"c"));
cost(1:Noofvariables)= c;

//---------------Constrain BV
BV = Noofvariables+1:size(A,2)-1;
// ---------------Calculate ZjCj
ZjCj = cost(BV)*A - cost;
//disp(ZjCj);
```

```
ZCj =[ZjCj;A];
//disp(ZCj);

mprintf('\n =========== Simplex Table ============\n')
disp(['x' 'x1' 'x2' 's1' 's2' "Sol"], [ ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5),ZCj(1:3,6)]);
//simplex table start

// check any negative value
k=0
for i= 1:size(ZjCj,2)
if (ZjCj(i) <0) then
    k=k+1
end
end
if k>0
   mprintf('\n    The current BFS is NOT Optimal \n')
end
disp("Old Basic Variable =  ");
disp(BV);
// Finding the Entering Variable
ZC = ZjCj(1:size(ZjCj,"c")-1);
[EnterCol,pvt_col] = min(ZC);
      mprintf('\n The Minimum element in Zj-Cj is %d
Corresponding to Column %d \n',EnterCol,pvt_col);

//Finding the Leaving Variable
sol= A(:,$);
Column =A(:,pvt_col);
z = size(Column,"r");
 j=0
for i=1:size(Column,"r")
   if Column(i)<0
   j=j+1;
   end
end
// To check UNBOUNDED
```

```
//disp(j);
if j == z;
   mprintf(' LPP is UNBOUNDED. All entries <= 0 in
column  %d \n',pvt_col);
end

for i=1:size(Column,"r")
   if Column(i)>0
    ratio(i) =sol(i)./Column(i);
  else
    ratio(i)=%inf;
   end
end

[MinRatio,pvt_row] = min(ratio);
mprintf('\n The Minimum Ratio Corresponding to PIVOT
Row  %d \n',pvt_row);
//n=pvt_row
disp([' LEAVING variable is'] ,[BV(pvt_row)]);
BV(pvt_row)=pvt_col;
disp(" New Basic Variable (BV)= ");
disp(BV);

//KEY ELEMENT
pvt_key = A(pvt_row,pvt_col);
//disp(pvt_key);

//UPDATE THE table for NEXT ITERATION
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

ZjCj = ZjCj-ZjCj(pvt_col).*A(pvt_row,:);
ZCj = [ZjCj;A];
mprintf('\n =========== Next Iteration ============\n')
```

```
disp(['x' 'x1' 'x2' 's1' 's2' "Sol"], [   ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5),ZCj(1:3,6)]);
end
BFS = zeros(1,size(A,2));
BFS(BV) = A(:,$);
BFS($) = sum(BFS.*cost);
//disp(BFS);
mprintf('\n  ============The BFS is =================
\n')
disp(['x' 'x1' 'x2' 's1' 's2' "Sol"], [   BFS(1,1) ,
BFS(1,2),BFS(1,3),BFS(1,4),BFS(1,5),BFS(1,6)]);
```

OUTPUT :

```
Scilab 6.1.1 Console

=========== Simplex Table ============

 "x"   "x1"   "x2"   "s1"   "s2"   "Sol"

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

   The current BFS is NOT Optimal

 "Old Basic Variable =   "

  4.    5.

The Minimum element in Zj-Cj is -3 Corresponding to Column 3

The Minimum Ratio Corresponding to PIVOT Row  1

 " LEAVING variable is"

  4.

 " New Basic Variable (BV)= "

  3.    5.

=========== Next Iteration ============

 "x"   "x1"   "x2"   "s1"   "s2"   "Sol"
```

```
  2.  -2.   0.   3.   0.   6.
  1.  -1.   1.   1.   0.   2.
 -1.   1.   1.   0.   1.   4.

============ Next Iteration =============

 "x"  "x1"  "x2"  "s1"  "s2"  "Sol"

  2.  -2.   0.   3.   0.   6.
  1.  -1.   1.   1.   0.   2.
 -2.   2.   0.  -1.   1.   2.

 ============The BFS is ================

 "x"  "x1"  "x2"  "s1"  "s2"  "Sol"

  0.   0.   2.   0.   2.   6.

-->
```

UNBOUNDED SOLUTIONS

Consider the linear program:

$$\begin{array}{llllll} \text{Maximize} & 2x_1 & +x_2 & & & \\ \text{Subject to:} & & & & & \\ & x_1 & -x_2 & \leq & 10 & (1) \\ & 2x_1 & -x_2 & \leq & 40 & (2) \end{array}$$

$$x_1,\, x_2 \geq 0\,.$$

SOURCE CODE :

```
clc;
//-----------INPUT PARAMETERS
Noofvariables=2;
c=[2 1 ]
info = [1 -1; 2 -1];
b =[ 10; 40];
n= size(info,"r");
s = eye(n,n);
A =[info s b];
//disp(A);
cost= zeros(1,size(A,"c"));
cost(1:Noofvariables)= c;

//---------------Constrain BV
BV = Noofvariables+1:size(A,2)-1;

// ---------------Calculate ZjCj
ZjCj = cost(BV)*A - cost;
//disp(ZjCj);
ZCj =[ZjCj;A];
//disp(ZCj);
mprintf('\n =========== Simplex Table ============\n')
```

```
disp(['x1' 'x2' 's1' 's2' "Sol"], [ ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5)]);
//simplex table start
k=0

// check any negative value
for i= 1:size(ZjCj,2)
 if (ZjCj(i) <0) then
     k=k+1
 end
if k>0
    mprintf('\n    The current BFS is NOT Optimal \n')

disp("Old Basic Variable =  ");
disp(BV);
// Finding the Entering Variable
ZC = ZjCj(1:size(ZjCj,"c")-1);
[EnterCol,pvt_col] = min(ZC);
      mprintf('\n The Minimum element in Zj-Cj is %d
Corresponding to Column %d \n',EnterCol,pvt_col);

//Finding the Leaving Variable
sol= A(:,$);
Column =A(:,pvt_col);
z = size(Column,"r");
 j=0
for i=1:size(Column,"r")
    if Column(i)<0
    j=j+1;
    end
end
// To check UNBOUNDED
  if j == z;
    mprintf(' LPP is UNBOUNDED. All entries <= 0 in
column  %d \n',pvt_col);

    break
```

```
  end;
for i=1:size(Column,"r")
   if Column(i)>0
    ratio(i) =sol(i)./Column(i);
  else
    ratio(i)=%inf;
   end
end

[MinRatio,pvt_row] = min(ratio);
mprintf('\n The Minimum Ratio Corresponding to PIVOT
Row  %d \n',pvt_row);
//n=pvt_row
disp([' LEAVING variable is'] ,[BV(pvt_row)]);
BV(pvt_row)=pvt_col;
disp(" New Basic Variable (BV)= ");
disp(BV);

//KEY ELEMENT
pvt_key = A(pvt_row,pvt_col);
//disp(pvt_key);

//UPDATE THE table for NEXT ITERATION
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

ZjCj = ZjCj-ZjCj(pvt_col).*A(pvt_row,:);
ZCj = [ZjCj;A];
mprintf('\n =========== Next Iteration ============\n')
disp(['x1' 'x2' 's1' 's2' "Sol"], [   ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5)]);
end
BFS = zeros(1,size(A,2));
BFS(BV) = A(:,$);
```

```
BFS($) = sum(BFS.*cost);
mprintf('\n  ============The BFS is ==================
\n')
disp(['x1' 'x2' 's1' 's2' "Sol"], [   BFS(1,1) ,
BFS(1,2),BFS(1,3),BFS(1,4),BFS(1,5)]);

end
end
```

OUTPUT :

```
Scilab 6.1.1 Console

=========== Simplex Table ============

 "x1"   "x2"   "s1"   "s2"   "Sol"

 -2.   -1.   0.    0.    0.
  1.   -1.   1.    0.    10.
  2.   -1.   0.    1.    40.

   The current BFS is NOT Optimal

 "Old Basic Variable =   "

  3.    4.

The Minimum element in Zj-Cj is -2 Corresponding to Column 1

The Minimum Ratio Corresponding to PIVOT Row   1

 " LEAVING variable is"

  3.

 " New Basic Variable (BV)= "

  1.    4.

=========== Next Iteration ============

 "x1"   "x2"   "s1"   "s2"   "Sol"

  0.   -3.   2.    0.    20.
  1.   -1.   1.    0.    10.
  2.   -1.   0.    1.    40.

=========== Next Iteration ============

 "x1"   "x2"   "s1"   "s2"   "Sol"
```

```
Scilab 6.1.1 Console

=========== Next Iteration ============

 "x1"  "x2"  "s1"  "s2"  "Sol"

  0.  -3.   2.   0.   20.
  1.  -1.   1.   0.   10.
  2.  -1.   0.   1.   40.

=========== Next Iteration ============

 "x1"  "x2"  "s1"  "s2"  "Sol"

  0.  -3.   2.   0.   20.
  1.  -1.   1.   0.   10.
  0.   1.  -2.   1.   20.

 ============The BFS is ================

 "x1"  "x2"  "s1"  "s2"  "Sol"

  10.   0.   0.   20.   20.

   The current BFS is NOT Optimal

 "Old Basic Variable =  "

  1.   4.

The Minimum element in Zj-Cj is -3 Corresponding to Column 2

The Minimum Ratio Corresponding to PIVOT Row  2

 " LEAVING variable is"

  4.

 " New Basic Variable (BV)= "

  1.   2.
```

```
=========== Next Iteration ============

 "x1"  "x2"  "s1"  "s2"  "Sol"

  0.   0.  -4.   3.   80.
  1.   0.  -1.   1.   30.
  0.   1.  -2.   1.   20.

=========== Next Iteration ============

 "x1"  "x2"  "s1"  "s2"  "Sol"

  0.   0.  -4.   3.   80.
  1.   0.  -1.   1.   30.
  0.   1.  -2.   1.   20.

 ============The BFS is ================

 "x1"  "x2"  "s1"  "s2"  "Sol"

  30.   20.   0.   0.   80.

   The current BFS is NOT Optimal

 "Old Basic Variable =  "

  1.   2.

The Minimum element in Zj-Cj is -4 Corresponding to Column 3
LPP is UNBOUNDED. All entries <= 0 in column  3

-->
```

MULTIPLE SOLUTIONS

Maximize $2000x_1 + 3000x_2$

subject to

$6x_1 + 9x_2 \leq 100$

$2x_1 + x_2 \leq 20$

$x_1, x_2 \geq 0$

SOURCE CODE :

```
clc;
//-----------INPUT PARAMETERS
Noofvariables=2;
```

```
c=[4 14]
info = [2  7; 7 2];
b =[ 21; 21];
n= size(info,"r");
s = eye(n,n);
A =[info s b];
//disp(A);
cost= zeros(1,size(A,"c"));
cost(1:Noofvariables)= c;

//---------------Constrain BV
BV = Noofvariables+1:size(A,2)-1;

// ---------------Calculate ZjCj
ZjCj = cost(BV)*A - cost;
//disp(ZjCj);
ZCj =[ZjCj;A];
//disp(ZCj)
mprintf('\n =========== Simplex Table ============\n')
disp(['x1' 'x2' 's1' 's2' "Sol"], [ ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5)]);
//simplex table start

// check any negative value
k=0
for i= 1:size(ZjCj,2)
if (ZjCj(i) <0) then
    k=k+1
end
end
if k>0
   mprintf('\n    The current BFS is NOT Optimal \n')
end
disp("Old Basic Variable =  ");
disp(BV);
// Finding the Entering Variable
ZC = ZjCj(1:size(ZjCj,"c")-1);
```

```
[EnterCol,pvt_col] = min(ZC);
      mprintf('\n The Minimum element in Zj-Cj is %d
Corresponding to Column %d \n',EnterCol,pvt_col);

//Finding the Leaving Variable
sol= A(:,$);
Column =A(:,pvt_col);
z = size(Column,"r");
 j=0
for i=1:size(Column,"r")
   if Column(i)<0
   j=j+1;
   end
end
// To check UNBOUNDED
//disp(j);
if j == z;
   mprintf(' LPP is UNBOUNDED. All entries <= 0 in
column  %d \n',pvt_col);
end

for i=1:size(Column,"r")
   if Column(i)>0
    ratio(i) =sol(i)./Column(i);
  else
    ratio(i)=%inf;
   end
end

[MinRatio,pvt_row] = min(ratio);
mprintf('\n The Minimum Ratio Corresponding to PIVOT
Row  %d \n',pvt_row);
//n=pvt_row
disp([' LEAVING variable is'] ,[BV(pvt_row)]);
BV(pvt_row)=pvt_col;
disp(" New Basic Variable (BV)= ");
disp(BV);
```

```
//KEY ELEMENT
pvt_key = A(pvt_row,pvt_col);
//disp(pvt_key);

//UPDATE THE table for NEXT ITERATION
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

ZjCj = ZjCj-ZjCj(pvt_col).*A(pvt_row,:);
ZCj = [ZjCj;A];
mprintf('\n =========== Next Iteration ============\n')
disp(['x1' 'x2' 's1' 's2' "Sol"], [   ZCj(1:3,1) ,
ZCj(1:3,2),ZCj(1:3,3),ZCj(1:3,4),ZCj(1:3,5)]);
end
BFS = zeros(1,size(A,2));
BFS(BV) = A(:,$);
BFS($) = sum(BFS.*cost);
//disp(BFS);
mprintf('\n  ============The BFS is =================
\n')
disp(['x1' 'x2' 's1' 's2' ], [   BFS(1,1) ,
BFS(1,2),BFS(1,3),BFS(1,4)]);
for i= 1:2
 if  BFS(1,i)== 0 then
     disp("Multiple Solutions as Zj-Cj value corresponding to non
basic variable is zero")
     else
 end
end
```

OUTPUT :

```
Scilab 6.1.1 Console

=========== Simplex Table ============

 "x1"   "x2"   "s1"   "s2"   "Sol"

 -4.   -14.   0.    0.    0.
  2.    7.    1.    0.    21.
  7.    2.    0.    1.    21.

   The current BFS is NOT Optimal

 "Old Basic Variable =   "

  3.    4.

The Minimum element in Zj-Cj is -14 Corresponding to Column 2

The Minimum Ratio Corresponding to PIVOT Row  1

 " LEAVING variable is"

  3.

 " New Basic Variable (BV)= "

  2.    4.

=========== Next Iteration ============

 "x1"   "x2"   "s1"   "s2"   "Sol"

  0.           0.   2.           0.   42.
  0.2857143    1.   0.1428571    0.   3.
  7.           2.   0.           1.   21.

=========== Next Iteration ============

 "x1"   "x2"   "s1"   "s2"   "Sol"
```

```
  0.           0.   2.           0.   42.
  0.2857143    1.   0.1428571    0.   3.
  6.4285714    0.  -0.2857143    1.   15.

 ===========The BFS is ===============

 "x1"  "x2"  "s1"  "s2"

  0.   3.   0.   15.

 "Multiple Solutions as Zj-Cj value corresponding to non basic variable is zero"

--> // -- 09/06/2022 13:00:55 -- //
```

EXPERIMENT 4

AIM : To determine local/relative optima of a given unconstrained problem.

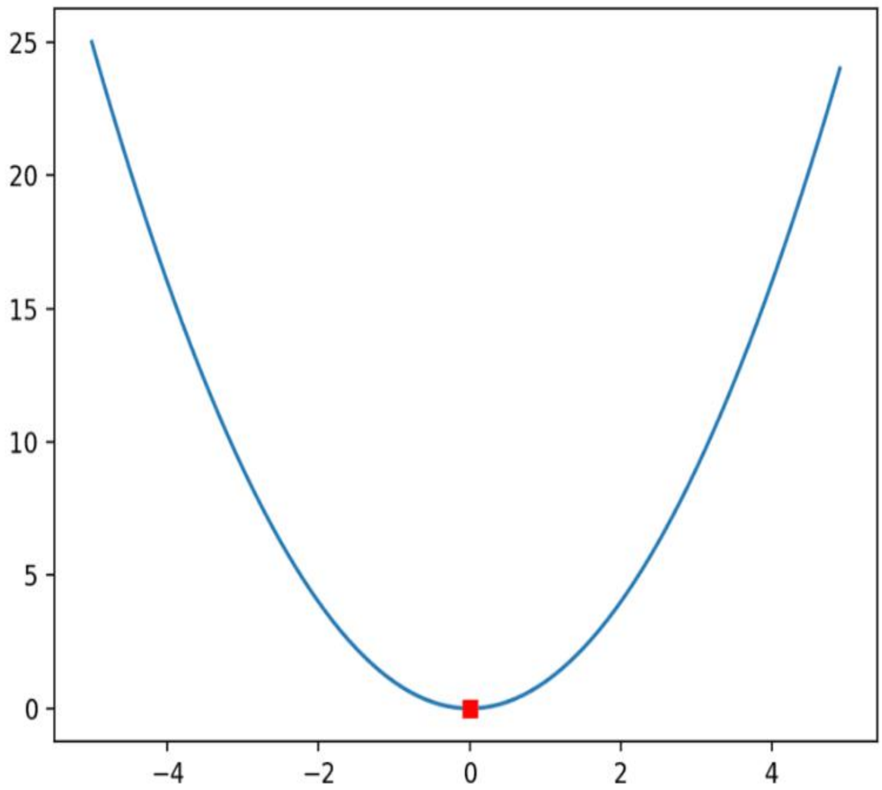
SOURCE CODE :

```
from numpy import arange
from matplotlib import pyplot

# objective function
def objective(x):
	return x**2.0

# define range for input
r_min, r_max = -5.0, 5.0
# sample input range uniformly at 0.1 increments
inputs = arange(r_min, r_max, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# define the known function optima
optima_x = 0.0
optima_y = objective(optima_x)
# draw the function optima as a red square
pyplot.plot([optima_x], [optima_y], 's', color='r')
# show the plot
pyplot.show()
```

OUTPUT :



EXPERIMENT 5

AIM : Test whether the given function is concave/convex .

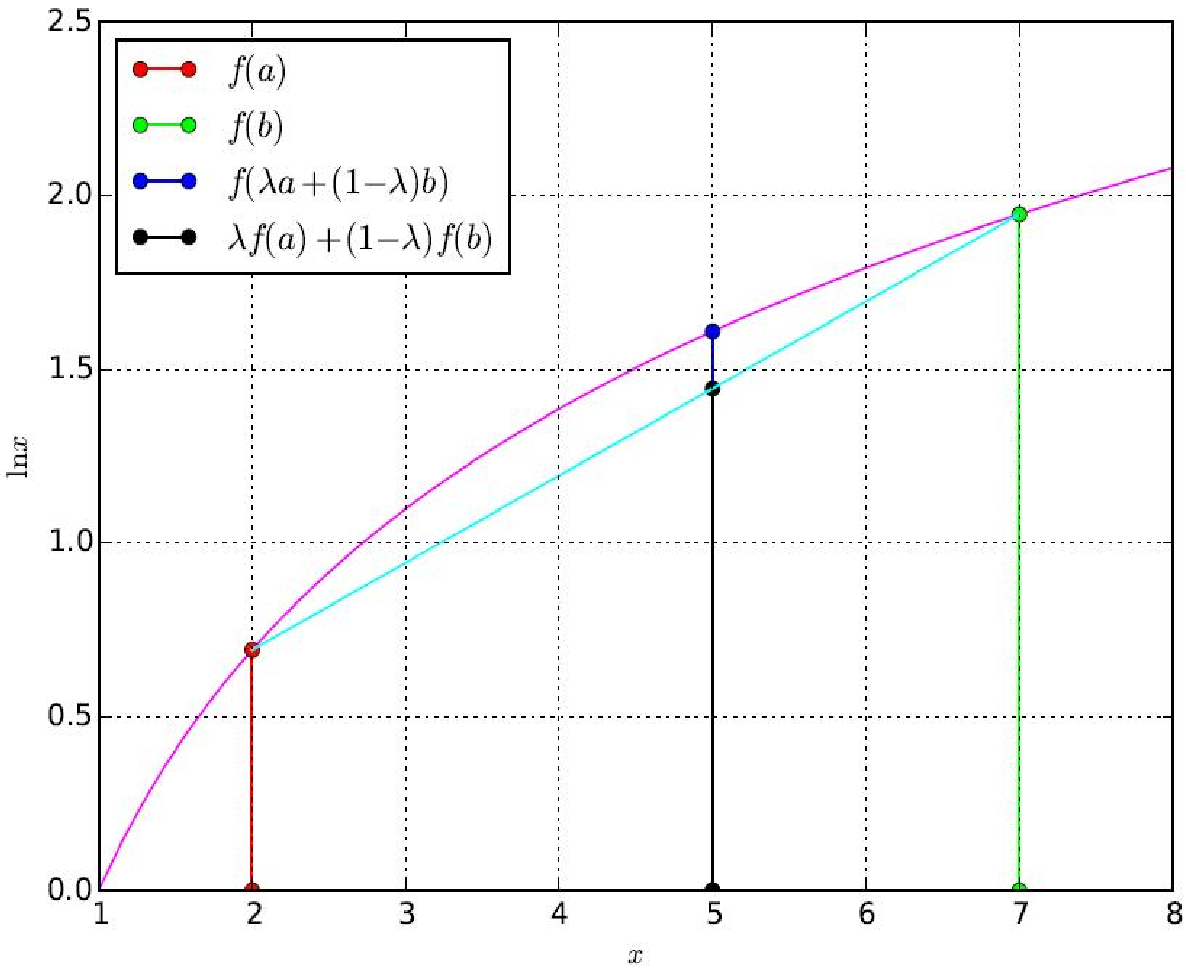
Function - *ln(x)*

SOURCE CODE :

```
import numpy as np
import matplotlib.pyplot as  plt
# Plotting log( x )
x = np.linspace( 1 , 8 , 5 0 )
# p o i n t s on t h e x a x i s
f=np.log( x ) # Ob j e c t i v e f u n c t i o n
plt.plot(x,f,color =( 1 , 0 , 1 ))
plt.grid( )
plt.xlabel('$x$ '')
plt.ylabel('$\lnx$' )
# Co n v e x i t y / Co n c a v i t y
a = 2
b = 7
lamda =0.4
c =lamda∗  a+(1−lamda)∗  b
f_a=np.log(a)
f_b=np.log(b)
f_c=np.log(c)
f_c_hat=lamda∗  f_a+(1−lamda)∗  f_b
# Plot commands
plt.plot([a,a],[0,f_a],color=(1,0,0),marker='o',label="$f(a)$")
plt.plot([b,b],[ 0 , f_b],color=(0,1,0),marker ='o',label=
"$f( b )$")
plt.plot([c,c],[0,f_c],color=(0,0,1),marker ='o',label= "
$f(\lambda a+(1−\lambda)b) $")
plt.plot ([c,c] , [0,f_c_hat ] , color=( 1 / 2 , 2 / 3 , 3 / 4), marker=
'o' ,label=" $\lambdaf(a)+(1−\lambda)f(b)$")
plt.plot([a,b], [f_a,f_b],color=(0 , 1 , 1))
plt.legend(loc=2)
```

```
# plt. save fig (  '. . / f i g s / 1 . 1 . eps ' )
plt.show( )
end
```

PLOT :



EXPERIMENT 6

AIM : Solution of optimization problems using Karush-Kuhn-Tucker conditions.

Problem 2.17. Solve

$$\min_{\mathbf{x}} \quad x_1 + x_2 \tag{2.53}$$

with the constraints

$$x_1^2 - x_1 + x_2^2 \leq 0 \tag{2.54}$$

where $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

THEORY :

In general for solving a convex optimization problem like using Lagrange Multipliers. These are called Karush-Kuhn-Tucker(KKT) conditions.

Using the method of Lagrange multipliers,

$\nabla \{ f(\mathrm{x}) + \mu g(\mathrm{x}) \} = 0, \mu \geq 0$

resulting in the equations

$2x_1 \mu - \mu + 1 = 0$

$2x^2 \mu + 1 = 0$

$\mathrm{x}^2{}_1 - x_1 + x_2 = 0$

so, $\mu = \sqrt{2}$.

Graphical solution: The constraint can be expressed

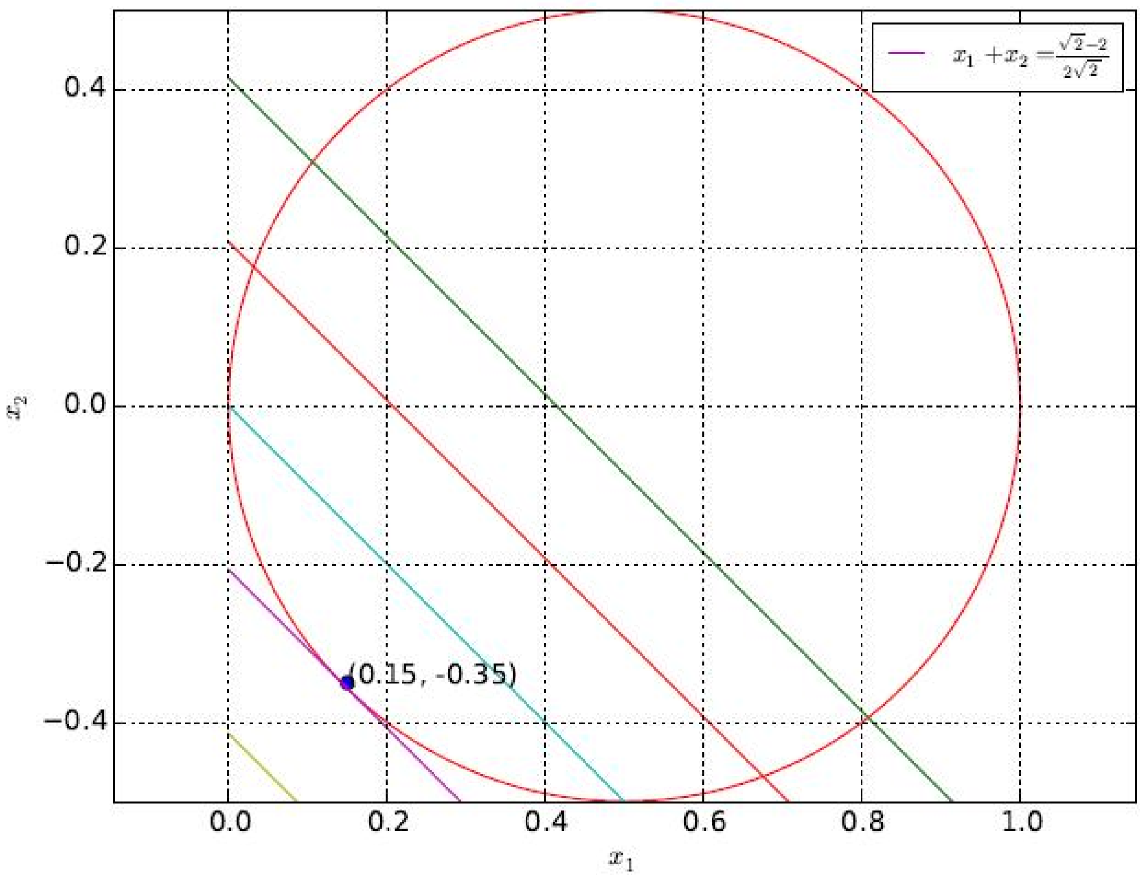
$x^2 1 - x_1 + x^2{}_1 <= 0$

SOURCE CODE :

```
import numpy a s np
import matplotlib.pyplotasplt
sol = np.zeros ((2,1) )
# Printing minimum
sol[0] =(np.s qrt(2)−1) / (2 *   np.sqrt( 2 ) )
sol[ 1] = −1/ (2*   np.sqr t (2 ) )
# Plotting the circle
circle = plt.Circle( (0.5 ,0) , 0.5 ,
color=’r’,fill =False)
fig , ax = plt.sub plots( )
ax .addartist(cir cle)

A = np.around ( sol[ 0 ] , decimals =2)
B = np.around ( sol [ 1 ] , decimals =2)
plt.plot (A,B, ’ o ‘)
for xy in zip (A,B) :
ax . annotate ( ‘(%s , %s )’ %xy , xy=xy , textcoords=’data ’ )
print (sol)
# Plotting the line
p =(sol[0]+ sol[1]) *  np . arange( −2 ,3)
x = np.linspace ( 0 , 1 ,100 )
na=np.newaxi s
x line = x [ : ,na ]
y line = p [ na , : ] − x[ : ,na ]
bx= plt . plot ( x line , y line , ’− ’ )
plt . axis ( ’ equal ’ )
plt . grid( )
plt . xlabel ( ’ $x 1$ ’ )
plt .ylabel ( ’$x 2$ ’ )
plt .ylim ( −0 .5 , 0. 5 )
plt . legend ( [ bx [ 3 ] ] , [ ’$x 1+x 2=\\ frac { \ sqr t { 2 } −2 }
{ 2 \ sqr t { 2 } } $ ’ ] ,
loc= ’ best ’ , prop ={ ’size ’ : 11 } )
plt.show ( )
```

OUTPUT :



$x_1 + x_2 = \frac{\sqrt{2}-2}{2\sqrt{2}}$
(0.15, -0.35)
0.4
0.2
0.0
−0.2
−0.4
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2

EXPERIMENT 7

AIM : Solution of Quadratic programming problem by Wolfe's method.

SOURCE CODE :

```
function [x, fval]=wolf(D, l, b, Mat, inq, minimize)
n = length(l);
m = length(b);

if ~isequal(size(Mat,1),m) || ~isequal(length(inq),m) ||
~isequal(size(D,1),size(D,2)) || ~isequal(size(D,1),n) ||
~isequal(size(Mat,2),n)
fprintf('\nError: Dimension mismatch!\n');
 return
end
if nargin < 4 || nargin > 6
   mprintf('\nError:Number of input arguments are
inappropriate!\n');
   return
end
if nargin < 5
   minimize = 0;
   inq = -ones(m,1);
elseif nargin < 6
   minimize = 0;
end
if minimize == 1
   l = -l;
   D = -D;
end
if min(spec(-D)) < 0 % Checking convexity of Hessian
   mprintf('\nError: Wolf method may not converge to global
optimum!\n');
   return
elseif (min(spec(-D)) == 0) && ~isempty(find(l,1))
```

```
    mprintf('\nError: Wolf method may not converge to global
optimum!\n');
    return
end
count = n;
for i = 1 : m
    if (inq(i) > 0)
        Mat(i,:) = -Mat(i,:);
        b(i) = -b(i);
    elseif (inq(i) == 0)
        count = count + 1;
        Mat(i,count) = -1;
        l(count) = 0;
        D(count,count) = 0;
    end
end
a = [-2*D Mat' -eye(count,count) zeros(count,m);Mat zeros(m,m
+ count) eye(m,m)];
d = [l;b];
for i = 1 : count + m
    if(d(i) < 0)
        d(i) = -d(i);
        a(i,:) = -a(i,:);
    end
end
cb = zeros(1,count + m);
bv = zeros(1,count + m);
nbv = (1 : 2 * (count + m));
c = zeros(1,2 * (count + m));
rem = zeros(1,count + m);
for i = 1 : count + m
    if(a(i,count + m + i) == -1)
        bv(i) = 2 * (count + m) + i;
        cb(i) = -1;
    elseif(a(i,count + m + i) == 1)
        rem(i)=count + m + i;
        bv(i) = count + m + i;
```

```
        cb(i) = 0;
    end
end
[h,j,k] = find(rem);
a(:,k) = [];
c(k) = [];
nbv(k) = [];
r = cb * a - c;
exitflg = 0;
iter = 0;
z = cb * d;
[w,y] = size(a);
opt = 0;

while(exitflg == 0)

    iter = iter + 1;
    mprintf('\n\n %d th tableau:\n',iter );
    mprintf('\n\t\t\tBV\t');disp(nbv);
    disp([bv' d a ;0 z r]);
    r_new = r;
    found = 0;
    while found == 0
        [u,v] = min(r_new);
        leave = 0;
        if ~(u < 0)
            if abs(z) > 10^-6
                mprintf('\nError: Wolf method fails to find
optimum!\n');
                exitflg = 1;
                found = 1;
            else
                mprintf('\nThe optimum has achieved!\n');
                exitflg = 1; opt = 1;
                found = 1;
            end
        else
```

```
ratio = 15;
check = 0;

for i = 1 : w
   if bv(i) <= 2 * (count +  m) && abs(bv(i) - nbv(v)) ==
count + m
      check = 1;
   end
end

if check == 0
   for i = 1 : w
      if a(i,v) > 0 && (d(i) / a(i,v)) < ratio
         ratio = d(i) / a(i,v);
         leave = i;
      end
   end

   mprintf('\nEntering Variable:'); disp(nbv(v));
   mprintf('\nLeaving Variable:'); disp(bv(leave));

   for i = 1 : w
      for j = 1 : y
         if i ~= leave && j ~= v
            a(i,j) = a(i,j) - a(i,v) * a(leave,j) / a(leave,v);
         end
      end
   end

   z = z - d(leave) * r(v) / a(leave,v);

   for j = 1 : y
      if j ~= v
         r(j) = r(j) - r(v) * a(leave,j) / a(leave,v);
         a(leave,j) = a(leave,j) / a(leave,v);
      end
   end
```

```
            for i = 1 : w
                if i ~= leave
                    d(i) = d(i) - a(i,v) * d(leave) / a(leave,v);
                    a(i,v) = -a(i,v) / a(leave,v);
                end
            end

            d(leave) = d(leave) / a(leave,v);
            a(leave,v) = 1 / a(leave,v);
            r(v) = -r(v) / a(leave,v);
            temp = nbv(v);
            nbv(v) = bv(leave);
            bv(leave) = temp;
            found = 1;
        elseif check == 1
            r_new(v) = 1;
        end
    end
  end
end

if opt == 1
  x = zeros(n,1);
  for i = 1 : w
    if bv(i) <= n
        x(bv(i)) = d(i);
    end
  end
  fval = x'*l+x'*D*x;
  if minimize == 1
    fval = -fval;
  end
end

end
```

OUTPUT :

Scilab 6.1.1 Console

File Edit Control Applications ?

Scilab 6.1.1 Console

```
--> exec('C:\Users\DELL\Desktop\simplex11.sce', -1)

--> D=[-3 1;1 -2];l=[2;3];b=[1;12];Mat=[1 1;3 4];inq=[1 -1];

-->

--> wolf(D,l,b,Mat,inq);


 1 th tableau:

                        BV
   2.   3.   4.   5.   6.   7.   8.

   9.   2.   -2.  -1.   3.  -1.   0.   0.   0.
   10.  3.    4.  -1.   4.   0.  -1.   0.   0.
   11.  1.    1.   0.   0.   0.   0.  -1.   0.
   8.   12.   4.   0.   0.   0.   0.   0.   1.
   0.  -6.   -3.   2.  -7.   1.   1.   1.   0.

Entering Variable:
   2.

Leaving Variable:
   10.


 2 th tableau:

                        BV
   10.   3.   4.   5.   6.   7.   8.

   9.    3.5    0.5   -1.5    5.  -1.  -0.5    0.   0.
   2.    0.75   0.25  -0.25   1.   0.  -0.25   0.   0.
   11.   0.25  -0.25   0.25  -1.   0.   0.25  -1.   0.
   8.    9.    -1.     1.    -4.   0.   1.     0.   1.
   0.   -3.75   12.    1.25  -4.   1.   0.25   1.   0.

Error: Wolf method fails to find optimum!
```