PRE MST LAB CODES

Shreeya Chatterji

GRAPHICAL METHOD

```
clf
clc
clear all
format short
c = [6, 11];
A = [2,1;1,2;0,1;1,0]
B = [104; 76; 0; 0]
const = [1;1] % 1 for less than -1 for greater than
objective = 1 % 1 for maximization
n = size(A,1);
x1 = 0 : 0.01 : max(B);

for i = 1:n-2
    y(i,:) = (B(i) - A(i,1)*x1) /A(i,2);
end

for i = 1:n-2
    y(i,:) = max (0, y(i,:));
    plot (x1, y(i,:), 'linewidth', 4)
    hold on
end
hold on

pt = [0; 0];
for i = 1:size(A,1)
    A1 = A(i,:);
    B1 = B(i,:);
    for j = i+1 : size(A,1)
        A2 = A(j,:);
```

① Defining the function

② finding y values

③ Plotting the lines

A1, A2, A3
B1, B2, B3
x3 → pt

④ finding the point of intersection

```
        B2 = B(j,:);
        A3 = [A1; A2];
        B3 = [B1; B2];
        X3 = A3\B3;
        if (X3>=0)
            pt = [pt X3];
        end
    end
end
```

```
X = pt';
X = unique(X, 'rows')
hold on
```

⑤ Inverse the pt matrix & keep only the unique points

```
x1 = X(:,1);
x2 = X(:,2);
for i = 1:n-2
    if (const(i)==1)
        ind = find(A(i,:)*x' > B(i))
        X(ind,:) = [];
    else
        ind = find(A(i,:)*x' < B(i))
        X(ind,:) = [];
    end
end
```

⑥ Keeping only the feasible solutions

```
obj_val = C*x';
if (objective==1)
    [value, ind] = max(obj_val);
else
```

⑦ finding the optimal solution & value

```
    [value, ind] = min(obj-val);
end
fprintf("Optimal value: %.f \n", value);
fprintf("Optimal point: %g %g \n", X(ind,:))
```

```
x = X(:,1);
y = X(:,2);
scatter(X(:,1), X(:,2), '*')
hold on
k = convhull(x,y);
fill(x(k), y(k), 'm')

xlim([0 max(x)+1])
ylim([0 max(y)+1])

xlabel('x-axis')
ylabel('y-axis')
title('feasible region of the LPP');
```

(8) Shade the feasible region & add labels as required

```
x = 0:0.1:max(B)
for z = 0:8:value
    y = (z - c(1)*x)/c(2);
    plot(x,y)
    hold on
    drawnow
    pause(0.0001)
end
hold on
```

(9) Checking by drawing lines

STEPS :-

1. Define the function → A, B, C, obj, const, n, x1
2. find y values → y(i,:) = (B(i) - A(i,1)*x1)/A(i,2)
3. Plot the lines → plot
4. find point of Intersection → A1, A2, A3, B1, B2, B3, X3 = A3\B3, Pt = [Pt X3]
5. Inverse the pt. matrix & keep only the unique points → X = pt' ; X = unique(X, 'rows')
6. Keep only feasible points ind = find(A(i,:)*x' >/< B(i))
7. Evaluate optimal sol^n & value c*x'
8. Shade feasible region & label scatter → convhull(x,y) → fill(x(k), y(k))
   xlim, ylim, xlabel, ylabel, 'm')
   title
9. Draw lines. plot

STANDARD LPP

```
clc
clear all
format short
```

① Define the function properly
② Change signs of the constr for -ve RHS
③ for ≥ constraints change the sign of 'S'
④ write with `array2table` & display

```
C = [2,-3,6];
A = [1,0,-3; 2,-8,3; 1,1,0];
B = [4;4;-7];
sign = [-1,1,-1];
n = size(A,1);
S = eye(n);
```

①

```
for i = 1:n
    if B(i) < 0
        A(i,:) = -A(i,:)
        B(i) = -B(i);
    end
end
```

②

```
indx = find(sign<0)
S(index,:) = -S(index,:);
```

③

④

```
objfun = array2table(C);
objfun.Properties.VariableNames(1:size(C,2)) = {'x1','x2','x3'};
Mat = [A S B]
const = array2table(Mat);
const.Properties.VariableNames(1:size(Mat,2)) = {'x1','x2','x3',
                                                 's1','s2','s3','RHS'};
disp("Objective fn:"); objfun;
disp("Standard LPP form"); const;
```

ALGEBRAIC METHOD

```
clc
clear all
format short
c = [5, -2, 0, 0];
A = [2, 5, 1, 0; 1, 1, 0, 1];
B = [8; 2];
[n, m] = size(A);
objective = -1; % -1 for minimize
nab = nchoosek(n, m);        <- ncm
t = nchoosek(1:n, m);        <- Generating an array which contains 1:n in m ways
sol = [ ]
if n >= m                    <- No. of variables must always be greater than no. of constr.
    for i = 1:nab
        y = zeroes(n, 1);
        x = (A(:, t(i, :)))\B;   <- for each index in (t) we find 'x'
        if all(x >= 0 & x ~= inf & x ~= -inf)
            y(t(i, :)) = x;      <- Storing point in y at 't'th index
            sol = [sol y];       <- Storing all feasible points in 'sol'
        end
    end
    disp("Solution:");
    disp(sol);
else
    error("No. of variables is less than no. of constraints");
end
if any(x == 0)
    fprintf('DEGENERATE SOL^N');
else
    fprintf("NON-DEGENERATE SOL^N");
```

① Defining fn

Checking Degeneracy.

```
Z = C*sol;
if (objective==1)
    [Zmax, Zindex] = max(Z);
else
    [Zmax, Zmin] = min(Z);
end
```

finding optimal soln

```
bfs = sol(:, Zindex);
[optimal_value] = [BFS' Zmax];
Optimal_bfs = array2table(Optimal_value);
Optimal_bfs.Properties.VariableNames(1:size(Optimal_bfs,2)) = {'x1','x2','s1','s2','val'};
disp(Optimal_bfs);
```

Printing the optimal point & value

Steps

(1) Define the function properly.

(2) Compute the number of basic solutions and extract the values we need to check `nab=nchoosek(n,m); t=nchoosek(1:n,m);`

(3) Construct the basic solution

- (i) Check if no of variables is more than the no of constraints ($n>=m$)
- (ii) Run a loop 'nab' times to compute the basic variables `X=(A(:,t(i,:)))\B;`
- (iii) Check feasibility ($X>=0$ & $X<=inf$ & $X<=-Inf$)
- (iv) Add the feasible soln to the solution matrix.
- (v) Display that solution.

(4) Check if degenerate.

(5) find the optimal solution by checking the minimum or maximum as demanded by the question.

(6) Extract the BFS from the zindex of the solution (sol) & convert into a matrix table with BFS & Zmap & print it.