LO Assignment
Group 20

Tushar Singh	2019394
Raj Kumar	2019084
Ishan Mehta	2019309

Question 1)

The code is in q1.py.
The user can fill the matrices A,b,C and then an object of class LPP is made using these matrices as inputs.

P = LPP(A,b,C)

The LPP returns an object P which is the linear programming problem -

Max $C^T X$,
Such that, $AX=b$ and $X>=0$

Number of equations = m
Number of variables = n

A is m x n matrix (coefficient matrix)
b is m x 1 vector (constant matrix)
C is n x 1 vector (cost matrix)
X is n x 1 vector (decision variables)

The solve() function takes in input the LPP ‘P’.
It first adds artificial variables to the LPP so that we can find the initial BFS.

We give the artificial variables a large cost M so that during optimization their value tends to 0.
Now the the LPP becomes ‘art_p’

Max $C_A^T X_A$
Such that, $A_A X_A = b$ and $X_A >= 0$

Where
$C_A^T = c_1, c_2, c_3 \ldots cn \ldots. -M, -M \ldots..-M$
$X_A = x_1, x_2 \ldots x, \ldots a_1, a_2 \ldots a_m$
$A_A = A + I_M$, here + mean concatenation

Also these values change for the new LPP ‘art_p’
Number of equations = m
Number of variables = n + m

A_A is m x (n +m) matrix (coefficient matrix)
b is m x 1 vector (constant matrix)
C_A is (n + m) x 1 vector (cost matrix)
X_A is (n + m) x 1 vector (decision variables)

And also we have an initial BFS in which all original decision variables are 0 and artificial variables are 1.

Now we pass the inputs art_p and its BFS into the function RSM.
This function solves the maximization problem art_p using revised simplex method.

In the revised simplex method we are not required to maintain the complete table as in the normal simplex method. We only need the following values -

B^{-1} = Inverse of basic variable matrix
Z = Current objective function = $C_B^T B^{-1} b$

W = used to find reduced costs for non basic variables = $C_B^T B^{-1}$
B’ = current values of basic variables = $B^{-1}b$

Using w we find the **reduced costs** of the non basic variables which is $C_B^T B^{-1} N - C_N^T$
If all these values are >=0 then we have reached the optimal point and exit the loop. Otherwise we select ‘nb_id’ (index of non basic variables with negative reduced cost).

After that we perform the **Minimum ratio test** to find the remaining basic variable.
These 2 values will give us a key row and a key column. We then pivot at the value at their intersection. Make this value 1 and then use it to make all entries in its column 0. By doing so the values z,w,b’,B^{-1} are updated automatically in the table. We then move back to the beginning of the loop.

Degeneracy - The code can report if we have found a degenerate point. This happens if there are more than n-m variables having 0 value in the optimal solution.

Unboundedness - The code can report unboundedness, this happens when during the minimum ratio test all the denominators are non positive. This means we can increase the non basic variable arbitrarily.

For question 2 and 3)
The above code is further modified as in q2.py and q3.py to be able to solve **Integer Linear Programming**.

For that we first solve the LPP without the constraints that the decision variables are integers. This can be done using the solve() function described above.After that we use **branch and bound method** to convert this real valued optimal solution into an integral optimal solution.
To do this we first pass the real valued optimal solution into the function solve_ILP() which then calls a recursive function branch().

branch() functions checks each variable and puts an upper and lower bound on it. This creates branches with slightly different LPPs. The branch() function continues to create these branches till we find an integral optimal solution.

Question 2)

Number of nodes = n
Number of edges = m

Lets define variables x_i for each ith edge as,

$X_i = 0$ if ith edge is not included in matching
$X_i = 1$ if ith edge is included in matching

This means total number of edges in matching is given by $\sum_{i=0}^{m} x_i$

We need to maximize this quantity such that each node is adjacent to at most 1 matched edge.

This means for all nodes V, $\sum x_i \leq 1$ where the summation is over all adjacent edges.

Then the maximum matching problem can be formulated as,

Max $\sum_{i=0}^{m} x_i$

Such that $X_I \in \{0,1\}$

For all nodes V, $\sum x_i \leq 1$, where the summation is over all adjacent edges.

Now we can solve this LPP using the solve() and solve_ILP() function described above

Question 3)

Number of nodes = n
Number of edges = m

Lets define variables x_i for each i^th node as,

$X_i = 0$ if i^th node is not included in cover
$X_i = 1$ if i^th node is included in cover

This means total number of nodes in cover is given by $\sum_{i=0}^{n} x_i$

We need to minimize this quantity such that each edge is adjacent to at least 1 covered node.

This means for all edges E, $\sum x_i \geq 1$ where the summation is over all adjacent nodes.

Then the minimum cover problem can be formulated as,

Min $\sum_{i=0}^{n} x_i$

Such that $X_I \in \{0,1\}$

For all nodes E, $\sum x_i \geq 1$, where the summation is over all adjacent nodes.

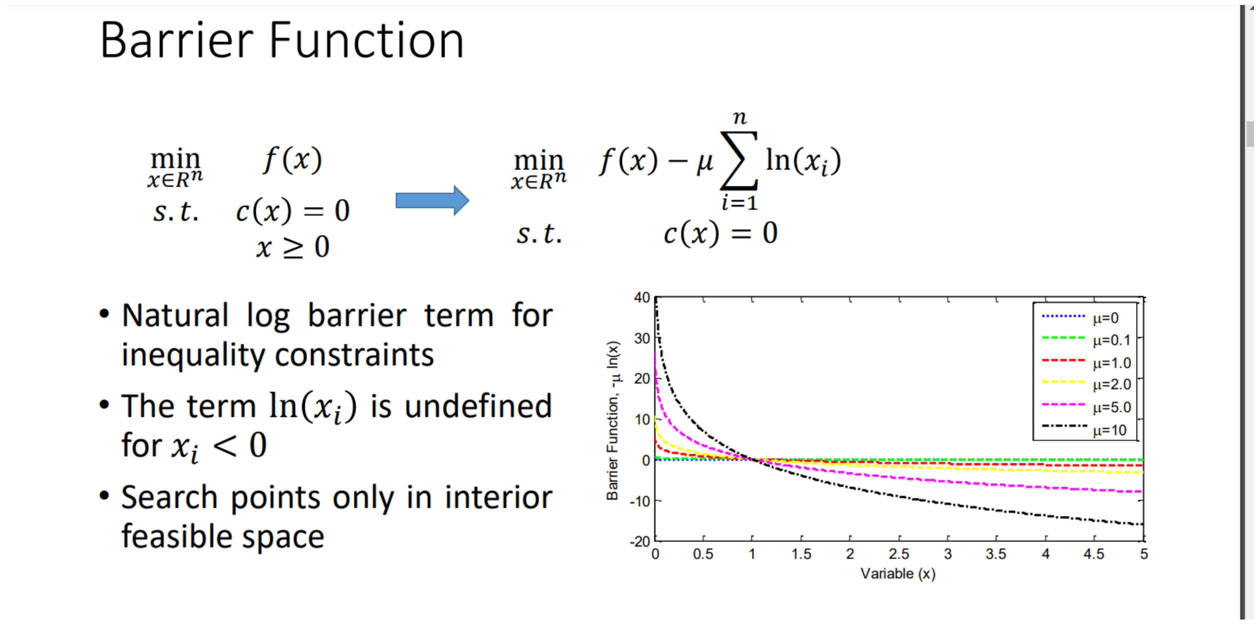
Now we can solve this LPP using the solve() and solve_ILP() function described above

We can see that the maximum edge matching problem and the minimum vertex cover problem are primal-dual pairs.

Question 4)

In question 4 we followed the following equations to derive the update rule using newton raphson method in interior point method(or ipopt)

These are the equations:



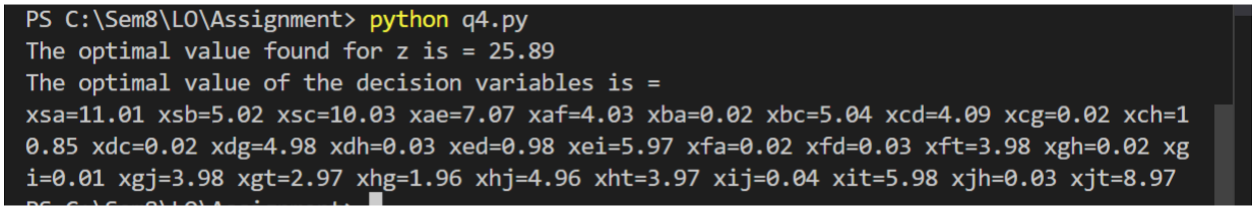
Barrier Function

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \min_{x \in R^n} \quad & f(x) - \mu \sum_{i=1}^{n} \ln(x_i) \\ s.t. \quad & c(x) = 0 \end{aligned}$$

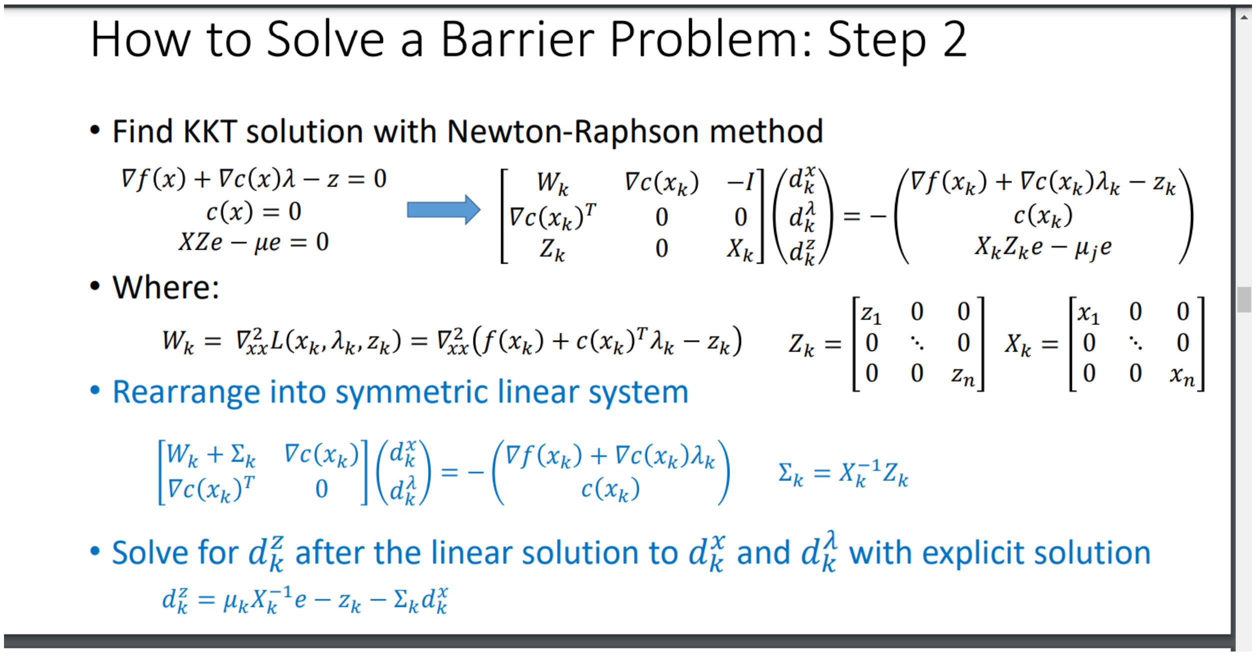
- Natural log barrier term for inequality constraints
- The term $\ln(x_i)$ is undefined for $x_i < 0$
- Search points only in interior feasible space

Here in first step the barrier function is used to add a penalty of -uln(x) to ensure that we get values in the

feasible region that is >=0 .
The inequality constraints are converted into slack and surplus variables so that at last we have all decision variables with >=0 values .



```
PS C:\Sem8\LO\Assignment> python q4.py
The optimal value found for z is = 25.89
The optimal value of the decision variables is =
xsa=11.01 xsb=5.02 xsc=10.03 xae=7.07 xaf=4.03 xba=0.02 xbc=5.04 xcd=4.09 xcg=0.02 xch=1
0.85 xdc=0.02 xdg=4.98 xdh=0.03 xed=0.98 xei=5.97 xfa=0.02 xfd=0.03 xft=3.98 xgh=0.02 xg
i=0.01 xgj=3.98 xgt=2.97 xhg=1.96 xhj=4.96 xht=3.97 xij=0.04 xit=5.98 xjh=0.03 xjt=8.97
```



How to Solve a Barrier Problem: Step 2

- Find KKT solution with Newton-Raphson method

$$\begin{aligned}\nabla f(x) + \nabla c(x)\lambda - z &= 0\\ c(x) &= 0\\ XZe - \mu e &= 0\end{aligned} \quad \Rightarrow \quad \begin{bmatrix} W_k & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)\lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix}$$

- Where:

$$W_k = \nabla_{xx}^2 L(x_k, \lambda_k, z_k) = \nabla_{xx}^2 \left(f(x_k) + c(x_k)^T \lambda_k - z_k\right) \qquad Z_k = \begin{bmatrix} z_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & z_n \end{bmatrix} \quad X_k = \begin{bmatrix} x_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & x_n \end{bmatrix}$$

- Rearrange into symmetric linear system

$$\begin{bmatrix} W_k + \Sigma_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)\lambda_k \\ c(x_k) \end{pmatrix} \qquad \Sigma_k = X_k^{-1} Z_k$$

- Solve for d_k^z after the linear solution to d_k^x and d_k^λ with explicit solution

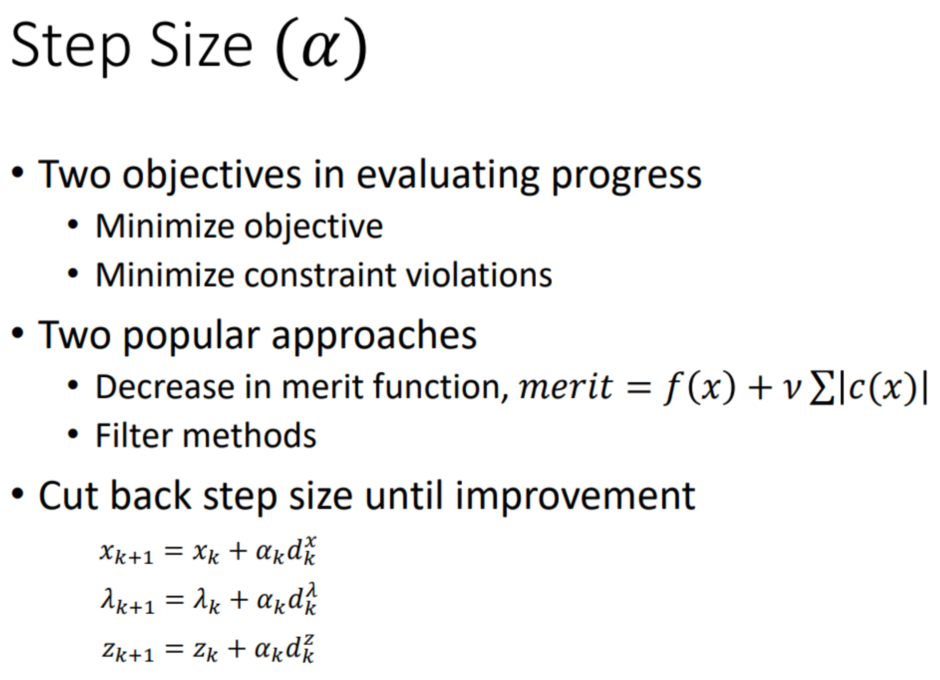
$$d_k^z = \mu_k X_k^{-1} e - z_k - \Sigma_k d_k^x$$

λ

Here these are the equations to determine dkx,dkλ and dkz where x denotes the decision variables z is 1/xi introduced to ensure non-negativity constraint on decision variables .

Now in our case the expression which represents double derivative of Lagrange(xk,λk,zk) w.r.t x is 0 expect at indexes (i,i) because there it will be equal to 1/(xi^2) because -1/(xi) will be derived again in that case. also delc(xk)*λk will be calculated where del(ck) is just the coefficients of the constraints for ex 2x1+3x2 constraint will have delc(xk) = $[2,3]^T$ and these will be summed for each constraint means λdel(c(xk)) = λ1del(c1(xk))+λ2*del(c2(xk)) uptill for nth constraint.
Now del(f(xk)) is just the coefficients of the objective function since the programming problem to be solved is linear in this case.
e vector is row of ones which is a vector of size (n_var) where n_var denotes number of variables.

After calculating all values mentioned in the above slide we calculate the value of xk+1, λk+1,zk+1 that means values of next iteration in next step using newton raphson method.



Step Size (α)

- Two objectives in evaluating progress
  - Minimize objective
  - Minimize constraint violations
- Two popular approaches
  - Decrease in merit function, $merit = f(x) + \nu \sum |c(x)|$
  - Filter methods
- Cut back step size until improvement

$$x_{k+1} = x_k + \alpha_k d_k^x$$
$$\lambda_{k+1} = \lambda_k + \alpha_k d_k^\lambda$$
$$z_{k+1} = z_k + \alpha_k d_k^z$$

Now in final step we just calculate the error estimations and when they are smaller than the upper cap of tolerance we end the iteration.

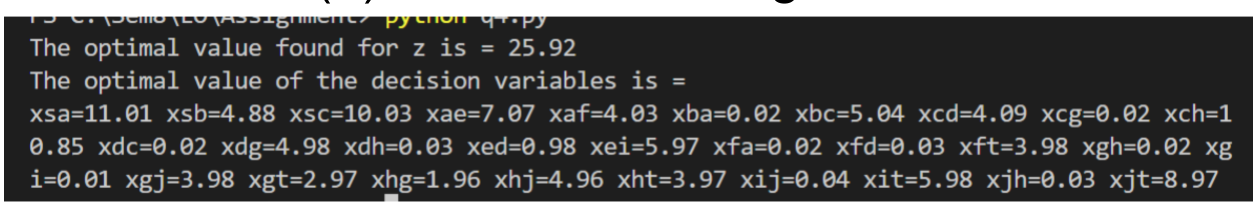
My full code for finding ipopt from scratch

```
import numpy as np
import pandas as pd
def interior_point_method(c,A,b,x0,mu=0.1,tol=0.5,max_iter=500):
    x,y = A.shape;
    n_var = y;m_equal=x;
    x0_prev = x0.reshape(n_var,1);
    z0_prev = mu / x0_prev;
    z0_prev = z0_prev.reshape((n_var,1));
    lambda_par = np.array([0]*m_equal);
    lambda_par = lambda_par.reshape((m_equal,1));
    val_min = 10**9;
    print("shapes x0_prev={} z0_prev={}
lambda_par={}".format(x0_prev.shape,z0_prev.shape,lambda_par.shape));
```

```
    for iter in range(max_iter):
        alpha = 1/(10);
        X_mat = np.diag(x0_prev.flatten());
        Z_mat = np.diag(z0_prev.flatten());

        A_derivative = np.zeros((n_var,m_equal));
        final_mat = np.zeros((n_var+m_equal,n_var+m_equal));
        for i in range(n_var):
            for j in range(m_equal):
                A_derivative[i][j] = A[j][i];
        W_k = np.zeros((n_var,n_var));
        for i in range(n_var):
            W_k[i][i] = 1/(x0_prev[i][0]**2);
        print("shapes X_mat={} Z_mat={} A_derivate={}
final_mat={}".format(X_mat.shape,Z_mat.shape,A_derivative.shape,final_mat.
shape))
        sigma_mat = np.linalg.inv((np.linalg.inv(X_mat))@Z_mat);
        new_mat= W_k+sigma_mat;
        A_t = A_derivative.T;
        for i in range(n_var):
            for j in range(n_var):
                final_mat[i][j] = new_mat[i][j];
        for i in range(0,n_var):
            for j in range(n_var,n_var+m_equal):
                final_mat[i][j] = A_derivative[i][j-n_var];
        for i in range(n_var,n_var+m_equal):
            for j in range(0,n_var):
                final_mat[i][j] = A_t[i-n_var][j];

        inv_mat = np.linalg.inv(final_mat);
        c_val = c;
        A_org = np.array([0.0]*n_var);
        for i in range(m_equal):
            A_org += lambda_par[i]*A[i,:];
        new_mat_rhs = np.array([0.0]*m_equal);
        for i in range(m_equal):
            val =
A[i,:].reshape((n_var,1)).T@x0_prev.reshape((n_var,1))+b[i];
            new_mat_rhs[i] = val;
        final_rhs = np.zeros((n_var+m_equal,1));
        for i in range(n_var+m_equal):
            if(i<n_var):
                final_rhs[i][0] = c_val[i]+A_org[i];
            else:
                final_rhs[i][0] = new_mat_rhs[i-n_var];
        result = inv_mat@(-final_rhs);
        result = result.reshape((n_var+m_equal,1));
        dx = np.zeros((n_var,1));
        dy = np.zeros((m_equal,1));
        for i in range(n_var+m_equal):
            if(i<n_var):
                dx[i][0] = result[i][0];
            else:
                dy[i-n_var][0] = result[i][0];
        # dz = np.zeros((n_var,1));
        ones_arr = np.ones((n_var,1));
        for i in range(m_equal):
            z0_prev[i][0] = 0;
        final_arr =
(mu*(np.linalg.inv(X_mat)@ones_arr))-(z0_prev.reshape((n_var,1)))-(sigma_m
at@dx);
        dz = final_arr.reshape((n_var,1));
        x0_new = x0_prev+alpha*dx;
        lambda_new = lambda_par+alpha*dy;
        z0_new = z0_prev+alpha*dz;
        # print("sum is here",((x0_new-x0_prev)>0).sum());

        # print("x0_new={} lambda0_new={}
z0_new={}".format(x0_new,lambda_new,z0_new));
        val_1=0;
        x0_prev = x0_new;
        lambda_par = lambda_new;
        z0_prev = z0_new;
        abb = 0;bcb = 0;
        for i in range(n_var):
            ab = abs(c_val[i]+A_org[i] - 1/x0_prev[i]);

            abb = max(ab,abb);
        ones_mat = np.ones((n_var,1));
        ones_mat = ones_mat.astype('float');
        for i in range(m_equal):
            bc = (A[i,:].reshape((n_var,1)).T@x0_prev.reshape((n_var,1)))
+b[i];
            bcb = max(abs(bc),bcb);
        X_mat_new = np.diag(x0_new.flatten());
        Z_mat_new = np.diag(z0_new.flatten());
        final_const = np.max(np.abs((X_mat_new@(Z_mat_new@ones_mat)) -
mu*(ones_mat)));








        if(abb<tol and bcb<tol and abs(final_const)<tol):
            break;
        else:
            print("Still here")
            print(abb,bcb,final_const);
        val_opt = val_min;


    print("optimal val={}".format(val_opt));
    print("decision variable values x0_values={}".format(x0_prev));
```

Ans-4(b)

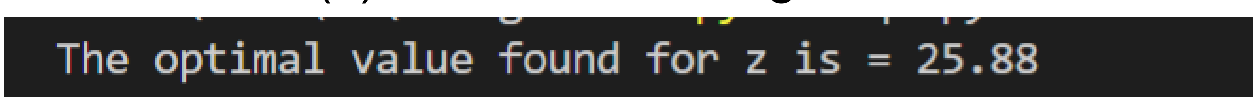
Solution for 4(b) for network 2 is given below:



```
PS C:\Sem8\LO\Assignment> python q4.py
The optimal value found for z is = 25.92
The optimal value of the decision variables is =
xsa=11.01 xsb=4.88 xsc=10.03 xae=7.07 xaf=4.03 xba=0.02 xbc=5.04 xcd=4.09 xcg=0.02 xch=1
0.85 xdc=0.02 xdg=4.98 xdh=0.03 xed=0.98 xei=5.97 xfa=0.02 xfd=0.03 xft=3.98 xgh=0.02 xg
i=0.01 xgj=3.98 xgt=2.97 xhg=1.96 xhj=4.96 xht=3.97 xij=0.04 xit=5.98 xjh=0.03 xjt=8.97
```

Ans-4(c)

Solution for 4(c) for network 2 is given below:



```
The optimal value found for z is = 25.88
```

Ans-5

Here in this problem we coded for flow network 2 max flow and min-cut LPP's and also coded for network 1 q2 LPP and its dual which is formed in q3.

For flow network 2:

Max-flow

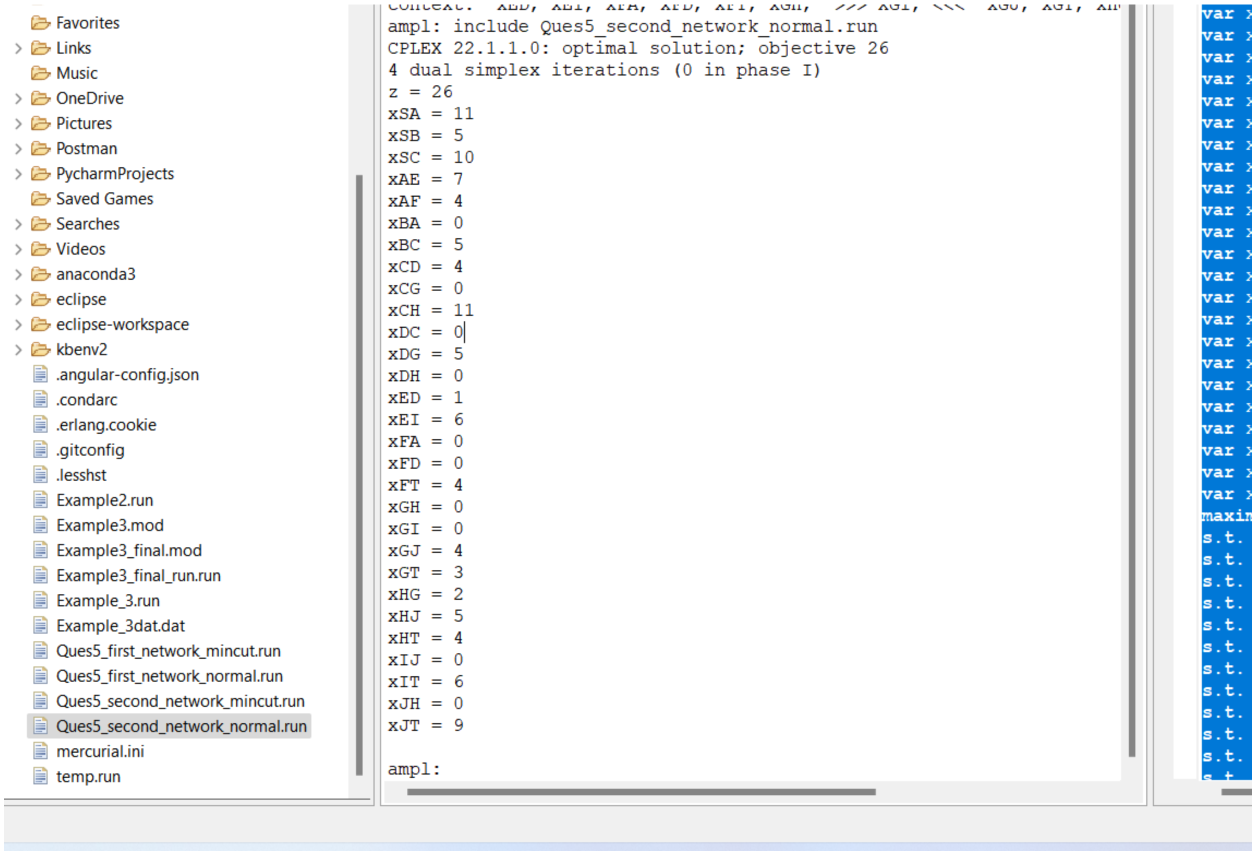
Code :

```
reset;
var xSA;
var xSB;
var xSC;
var xAE;
var xAF;
var xBA;
var xBC;
var xCD;
var xCG;
var xCH;
var xDC;
var xDG;
var xDH;
var xED;
var xEI;
var xFA;
var xFD;
var xFT;
var xGH;
var xGI;
var xGJ;
var xGT;
var xHG;
var xHJ;
var xHT;
var xIJ;
var xIT;
var xJH;
var xJT;
maximize z: xSA + xSB + xSC;
s.t. c1: xSA + xBA + xFA - xAE - xAF = 0;
s.t. c2: xSB - xBC = 0;
s.t. c3: xSC + xBC + xDC - xCD - xCG - xCH = 0;
s.t. c4: xCD + xED + xFD - xDC - xDG - xDH = 0;
s.t. c5: xAE - xED - xEI = 0;
s.t. c6: xAF - xFA - xFD - xFT = 0;
```

```
s.t. c7: xCG + xDG + xHG - xGH - xGI - xGJ - xGT =
0;
s.t. c8: xCH + xDH + xGH + xJH - xHG - xHJ - xHT =
0;
s.t. c9: xEI + xGI - xIJ - xIT = 0;
s.t. c10: xGJ + xHJ + xIJ - xJH - xJT = 0;
s.t. c12: 0 <= xSA <= 11;
s.t. c13: 0 <= xSB <= 15;
s.t. c14: 0 <= xSC <= 10;
s.t. c15: 0 <= xAE <= 18;
s.t. c16: 0 <= xAF <= 4;
s.t. c17: 0 <= xBA <= 3;
s.t. c18: 0 <= xBC <= 5;
s.t. c19: 0 <= xCD <= 6;
s.t. c20: 0 <= xCG <= 3;
s.t. c21: 0 <= xCH <= 11;
s.t. c22: 0 <= xDC <= 4;
s.t. c23: 0 <= xDG <= 17;
s.t. c24: 0 <= xDH <= 6;
s.t. c25: 0 <= xED <= 3;
s.t. c26: 0 <= xEI <= 13;
s.t. c27: 0 <= xFA <= 12;
s.t. c28: 0 <= xFD <= 4;
s.t. c29: 0 <= xFT <= 21;
```

```
s.t. c30: 0 <= xGH <= 4;
s.t. c31: 0 <= xGI <= 9;
s.t. c32: 0 <= xGJ <= 4;
s.t. c33: 0 <= xGT <= 3;
s.t. c34: 0 <= xHG <= 4;
s.t. c35: 0 <= xHJ <= 5;
s.t. c36: 0 <= xHT <= 4;
s.t. c37: 0 <= xIJ <= 7;
s.t. c38: 0 <= xIT <= 9;
s.t. c39: 0 <= xJH <= 2;
s.t. c40: 0 <= xJT <= 15;
option solver cplex;
solve;
display z, xSA , xSB, xSC, xAE, xAF, xBA, xBC,
xCD, xCG, xCH, xDC, xDG, xDH,
 xED, xEI, xFA, xFD, xFT, xGH, xGI, xGJ, xGT, xHG,
xHJ, xHT, xIJ, xIT, xJH, xJT;
```

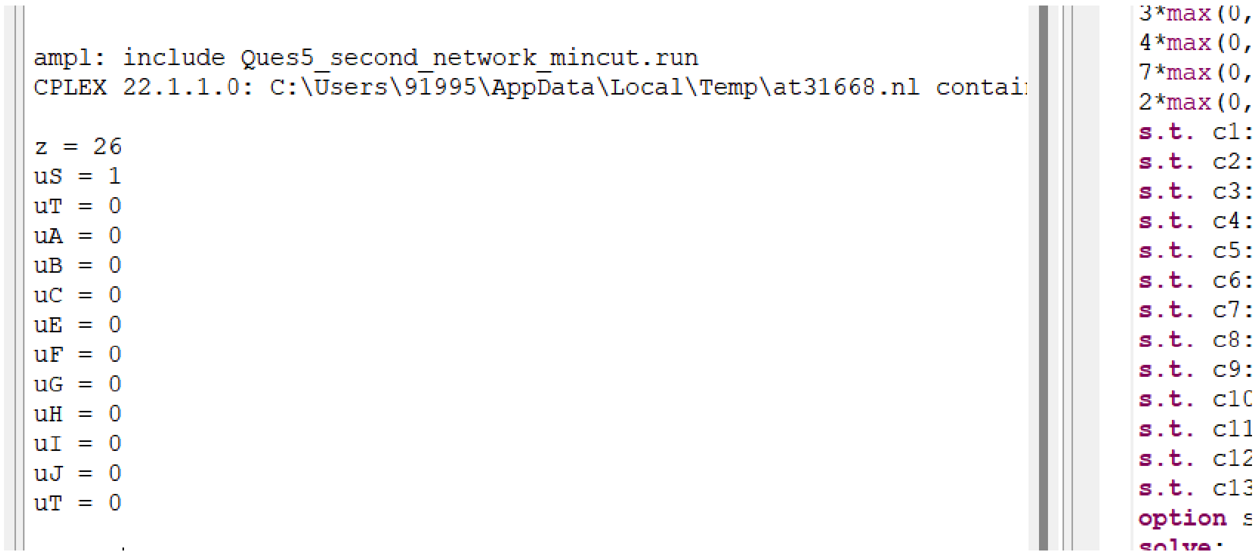
soln:



```
ampl: include Ques5_second_network_normal.run
CPLEX 22.1.1.0: optimal solution; objective 26
4 dual simplex iterations (0 in phase I)
z = 26
xSA = 11
xSB = 5
xSC = 10
xAE = 7
xAF = 4
xBA = 0
xBC = 5
xCD = 4
xCG = 0
xCH = 11
xDC = 0
xDG = 5
xDH = 0
xED = 1
xEI = 6
xFA = 0
xFD = 0
xFT = 4
xGH = 0
xGI = 0
xGJ = 4
xGT = 3
xHG = 2
xHJ = 5
xHT = 4
xIJ = 0
xIT = 6
xJH = 0
xJT = 9

ampl:
```

b)

Second_mincut



```
ampl: include Ques5_second_network_mincut.run
CPLEX 22.1.1.0: C:\Users\91995\AppData\Local\Temp\at31668.nl contai

z = 26
uS = 1
uT = 0
uA = 0
uB = 0
uC = 0
uE = 0
uF = 0
uG = 0
uH = 0
uI = 0
uJ = 0
uT = 0
```

```
3*max(0,
4*max(0,
7*max(0,
2*max(0,
s.t. c1:
s.t. c2:
s.t. c3:
s.t. c4:
s.t. c5:
s.t. c6:
s.t. c7:
s.t. c8:
s.t. c9:
s.t. c10
s.t. c11
s.t. c12
s.t. c13
option s
solve;
```

Code:

```
reset;
var uS;
var uA;
var uB;
var uC;
var uD;
var uE;
var uF;
var uG;
var uH;
var uI;
var uJ;
var uT;
minimize z: 11*max(0, uS-uA) + 15*max(0, uS-uB) +
0*max(0, uS-uC) +
18*max(0, uA-uE) + 4*max(0, uA-uF) +
3*max(0, uB-uA) + 5*max(0, uB-uC) +
6*max(0, uC-uD) + 3*max(0, uC-uG) + 11*max(0,
uC-uH)
+
4*max(0, uD-uC) + 17*max(0, uD-uG) + 6*max(0,
uD-uH)
```

```
+
3*max(0, uE-uD) + 13*max(0, uE-uI) +
12*max(0, uF-uA) + 4*max(0, uF-uD) + 21*max(0,
uF-uT)
+
4*max(0, uG-uH) + 9*max(0, uG-uI) + 4*max(0,
uG-uJ) +
3*max(0, uG-uT) +
4*max(0, uH-uG) + 5*max(0, uH-uJ) + 4*max(0,
uH-uT) +
7*max(0, uI-uJ) + 9*max(0, uI-uT) +
2*max(0, uJ-uH) + 15*max(0, uJ-uT);
s.t. c1: uS = 1;
s.t. c2: uT = 0;
s.t. c3: uA >= 0;
s.t. c4: uB >= 0;
s.t. c5: uC >= 0;
s.t. c6: uD >= 0;
s.t. c7: uE >= 0;
s.t. c8: uF >= 0;
s.t. c9: uG >= 0;
s.t. c10: uH >= 0;
s.t. c11: uI >= 0;
s.t. c12: uJ >= 0;
```

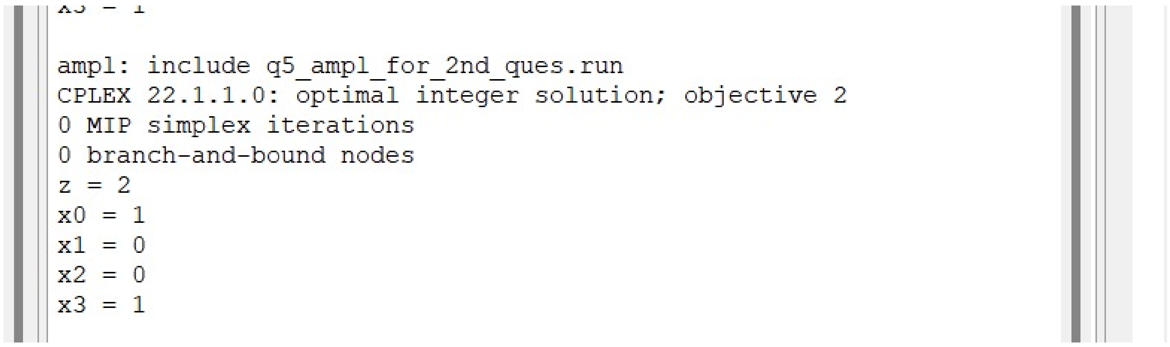
```
s.t. c13: uI >= 0;
option solver cplex;
solve;
display z;
```

c)

q2 LPP problem.

```
reset;

var x0 integer;
var x1 integer;
var x2 integer;
var x3 integer;

maximize z: x0 + x1 + x2 + x3;

s.t. c1: x0 <= 1;
s.t. c2: x0 + x1 + x2 <= 1;
s.t. c3: x1 + x3 <= 1;
s.t. c4: x2 + x3 <= 1;
s.t. c5: x0 >= 0;
s.t. c6: x1 >= 0;
s.t. c7: x2 >= 0;
s.t. c8: x3 >= 0;
```

option solver cplex;
solve;
display z, x0, x1, x2, x3;



```
ampl: include q5_ampl_for_2nd_ques.run
CPLEX 22.1.1.0: optimal integer solution; objective 2
0 MIP simplex iterations
0 branch-and-bound nodes
z = 2
x0 = 1
x1 = 0
x2 = 0
x3 = 1
```

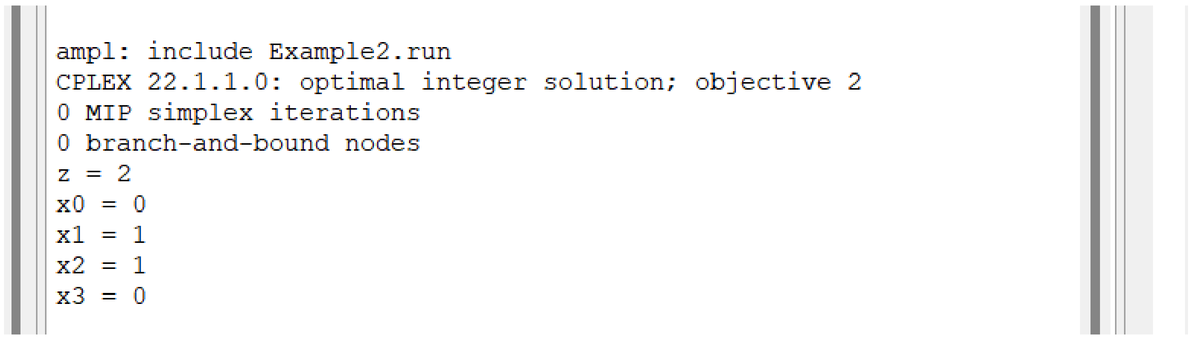
5(d)
q3 LPP problem:
reset;

var x0 integer;
var x1 integer;
var x2 integer;
var x3 integer;

minimize z: x0 + x1 + x2 + x3;

s.t. c1: x0 + x1 >= 1;
s.t. c2: x1 + x2 >= 1;
s.t. c3: x2 + x3 >= 1;

```
s.t. c4: x1 + x3 >= 1;
s.t. c5: x0 >= 0;
s.t. c6: x1 >= 0;
s.t. c7: x2 >= 0;
s.t. c8: x3 >= 0;

option solver cplex;
solve;
display z, x0, x1, x2, x3;
```



```
ampl: include Example2.run
CPLEX 22.1.1.0: optimal integer solution; objective 2
0 MIP simplex iterations
0 branch-and-bound nodes
z = 2
x0 = 0
x1 = 1
x2 = 1
x3 = 0
```