

Name: Siddhesh Vengurlekar

Roll No.: 505

Operations Research (Paper III)
MSc. (Computer Science) Semester III 2022-23

Index

Sr No.	Date	Title	Page No.	Sign
1		Graphical Method	2	
2		Simplex Method for 2 variables	5	
3		Simplex Method for 3 variables	7	
4		Simplex Method for equality constraint	9	
5		Big M Method	11	
6		Resource Allocation problem	13	
7		Infeasible solution	15	
8		Dual Simplex Method	17	
9		Transportation Method	19	
10		Assignment Method	21	

Practical 1

Aim: Use graphical method to solve the following LPP:

$$\begin{aligned} \text{Max } Z &= 3x + 5y \\ \text{w.r.t.} \\ x + 2y &\leq 2000, \\ x + y &\leq 1500, \\ y &\leq 600, \\ x, y &\geq 0 \end{aligned}$$

Source Code:

```
require(lpSolve)

C ← c(3, 5)

A ← matrix(c(1, 2,
             1, 1,
             0, 1), nrow = 3, byrow = T)

B ← c(2000, 1500, 600)

constraint_direction ← c("≤", "≤", "≤")

plot.new()
plot.window(xlim = c(0, 2000), ylim = c(0, 2000))
axis(1)
axis(2)
title(main = "LPP using graphical method", xlab = "X-axis", ylab = "Y-
axis")
box()

segments(x0 = 2000, y0 = 0, x1 = 0, y1 = 1000, col = "green")
segments(x0 = 1500, y0 = 0, x1 = 0, y1 = 1500, col = "red")
segments(x0 = 0, y0 = 0, x1 = 600, y1 = 0, col = "blue")

z ← lp(direction = "max",
```

```

objective.in = C,
const.mat = A,
const.dir = constraint_direction,
const.rhs = B,
all.int = T
)

```

```
print(z$status)
```

```

best_sol ← z$solution
names(best_sol) ← c("x1", "x2")
print(paste("Total cost: ", z$objval, sep = ""))

```

Output:

```

> # Use graphical method to solve the following LPP:
> # Max z = 3x + 5y
> # w.r.t
> #  $x + 2y \leq 2000$ ,
> #  $x + y \leq 1500$ ,
> #  $y \leq 600$ ,
> #  $x, y \geq 0$ 
>
> require(lpSolve)
>
> C ← c(3, 5)
>
> A ← matrix(c(1, 2,
+             1, 1,
+             0, 1), nrow = 3, byrow = T)
>
> B ← c(2000, 1500, 600)
>
> constraint_direction ← c("<=", "<=", "<=")
>
> plot.new()
> plot.window(xlim = c(0, 2000), ylim = c(0, 2000))

```

```

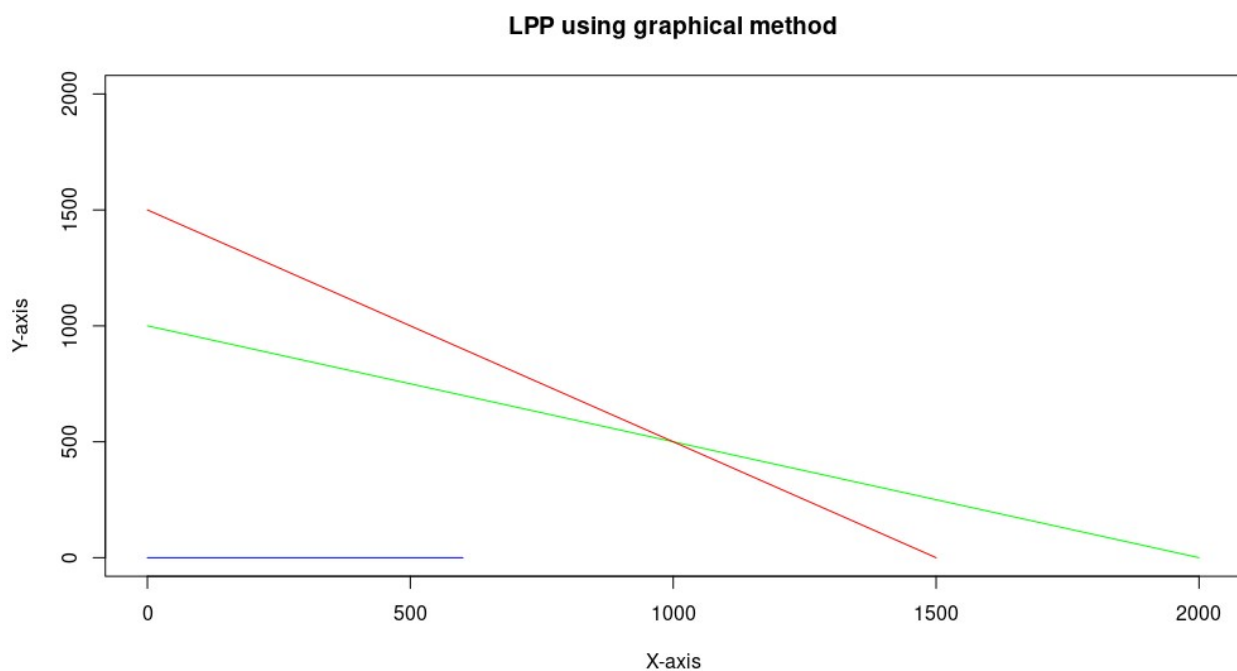
> axis(1)
> axis(2)
> title(main = "LPP using graphical method", xlab = "X-axis", ylab = "Y-axis")
> box()
>
> segments(x0 = 2000, y0 = 0, x1 = 0, y1 = 1000, col = "green")
> segments(x0 = 1500, y0 = 0, x1 = 0, y1 = 1500, col = "red")
> segments(x0 = 0, y0 = 0, x1 = 600, y1 = 0, col = "blue")
>
> z <- lp(direction = "max",
+         objective.in = C,
+         const.mat = A,
+         const.dir = constraint_direction,
+         const.rhs = B,
+         all.int = T
+         )
>
> print(z$status)
[1] 0
>
> best_sol <- z$solution
> names(best_sol) <- c("x1", "x2")

```

```

> best_sol <- z$solution
> names(best_sol) <- c("x1", "x2")
> print(paste("Total cost: ", z$objval, sep = ""))
[1] "Total cost: 5500"
>

```



Practical 2

Aim: Use simplex method to solve the following LPP:

$$\begin{aligned}\text{Max } Z &= 3x + 2y \\ \text{w.r.t.} \\ x + y &\leq 4, \\ x - y &\leq 2, \\ x, y &\geq 0\end{aligned}$$

Source Code:

```
from scipy.optimize import linprog

obj = [-3, -2]
lhs_ineq = [[1, 1], [1, -1]]
In[1]: rhs_ineq = [4, 2]

bound = [(0, float("inf")), (0, float("inf"))]

z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
In[2]:          bounds = bound, method = "revised simplex")

z

In[3]: print(z.fun)
       print(z.success)
       print(z.x)
```

Output:

```
In [1]: 1 from scipy.optimize import linprog
        2
        3 obj = [-3, -2]
        4 lhs_ineq = [[1, 1],
        5              [1, -1]]
        6
        7 rhs_ineq = [4,
        8              2]
        9
       10 bound = [(0, float("inf")),
       11              (0, float("inf"))]
```

```
In [2]: 1 z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
        2              bounds = bound, method = "revised simplex")
        3
        4 z
```

```
Out[2]:      con: array([], dtype=float64)
          fun: -11.0
          message: 'Optimization terminated successfully.'
          nit: 2
          slack: array([0., 0.])
          status: 0
          success: True
          x: array([3., 1.])
```

```
In [3]: 1 print(z.fun)
        2 print(z.success)
        3 print(z.x)
```

```
-11.0
True
[3. 1.]
```

Practical 3

Aim: Use simplex method to solve the following LPP:

$$\text{Min } Z = x_1 - 3x_2 + 2x_3$$

w.r.t

$$3x_1 - x_2 + 3x_3 \leq 7,$$

$$-2x_1 + 4x_2 \leq 12,$$

$$-4x_1 + 3x_2 + 8x_3 \leq 10,$$

$$x_1, x_2, x_3 \geq 0$$

Source Code:

```
from scipy.optimize import linprog
```

```
obj = [1, -3, 2]
```

```
lhs_ineq = [[3, -1, 3],  
            [-2, 4, 0],  
            [-4, 3, 8]]
```

```
In [1]: rhs_ineq = [7,  
                   12,  
                   10]
```

```
bound = [(0, float("inf")),  
         (0, float("inf")),  
         (0, float("inf"))]
```

```
In [2]: z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,  
                   bounds = bound, method = "revised simplex")
```

z

Output:

```
In [1]: 1 from scipy.optimize import linprog
2
3 obj = [1, -3, 2]
4
5 lhs_ineq = [[3, -1, 3],
6             [-2, 4, 0],
7             [-4, 3, 8]]
8
9 rhs_ineq = [7,
10            12,
11            10]
12
13 bound = [(0, float("inf")),
14          (0, float("inf")),
15          (0, float("inf"))]

In [2]: 1 z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
2             bounds = bound, method = "revised simplex")
3
4 z

Out[2]: con: array([], dtype=float64)
fun: -11.0
message: 'Optimization terminated successfully.'
nit: 2
slack: array([ 0.,  0., 11.])
status: 0
success: True
x: array([4., 5., 0.])
```


Practical 4

Aim: Use simplex method to solve the following LPP:

$$\begin{aligned}\text{Max } Z &= x + 2y \\ \text{w.r.t.} \\ 2x + y &\leq 20, \\ -4x + 5y &\leq 10, \\ -x + 2y &\geq -2, \\ -x + 5y &= 15, \\ x, y &\geq 0\end{aligned}$$

Source code:

```
from scipy.optimize import linprog

obj = [-1, -2]

lhs_ineq = [[2, 1],
            [-4, 5],
            [1, -2]]

In [1]: rhs_ineq = [20,
                    10,
                    2]

lhs_eq = [[-1, 5]]
rhs_eq = [15]

bound = [(0, float("inf")),
         (0, float("inf"))]

In [2]: z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
                    A_eq = lhs_eq, b_eq = rhs_eq,
                    bounds = bound, method = "revised simplex")

Z
```

Output:

```
In [1]: 1 from scipy.optimize import linprog
2
3 obj = [-1, -2]
4
5 lhs_ineq = [[2, 1],
6             [-4, 5],
7             [1, -2]]
8
9 rhs_ineq = [20,
10            10,
11            2]
12
13 lhs_eq = [[-1, 5]]
14 rhs_eq = [15]
15
16 bound = [(0, float("inf")),
17           (0, float("inf"))]

In [2]: 1 z = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
2             A_eq = lhs_eq, b_eq = rhs_eq,
3             bounds = bound, method = "revised simplex")
4
5 z

Out[2]: con: array([0.])
fun: -16.818181818181817
message: 'Optimization terminated successfully.'
nit: 3
slack: array([ 0.          , 18.18181818,  3.36363636])
status: 0
success: True
x: array([7.72727273, 4.54545455])
```

Practical 5

Aim: Use Big M method to solve the following LPP:

$$\begin{aligned}\text{Min } Z &= 4x_1 + x_2 \\ \text{w.r.t.} \\ 3x_1 + 4x_2 &\geq 12, \\ x_1 + 5x_2 &\geq 15, \\ x_1, x_2 &\geq 0\end{aligned}$$

Source code:

```
from scipy.optimize import linprog

obj = [4, 1]
lhs_ineq = [[-3, -4],
            [-1, -5]]

In [1]: rhs_ineq = [-20,
                  -15]

bound = [(0, float("inf")),
         (0, float("inf"))]

opt = linprog(c=obj,A_ub=lhs_ineq,b_ub=rhs_ineq,
              bounds=bound,method="interior-point")

In [2]: opt
```

Output:

```
In [1]: 1 from scipy.optimize import linprog
        2
        3 obj = [4, 1]
        4 lhs_ineq = [[-3, -4],
        5               [-1, -5]]
        6
        7 rhs_ineq = [-20,
        8               -15]
        9
       10 bound = [(0, float("inf")),
       11             (0, float("inf"))]
```

```
In [2]: 1 opt =linprog(c=obj,A_ub=lhs_ineq,b_ub=rhs_ineq,
        2               bounds=bound,method="interior-point")
        3
        4 opt
```

```
Out[2]:      con: array([], dtype=float64)
         fun: 5.0000000002364455
         message: 'Optimization terminated successfully.'
         nit: 5
         slack: array([1.64270375e-10, 1.00000000e+01])
         status: 0
         success: True
         x: array([6.01160437e-11, 5.00000000e+00])
```

Practical 6

Aim: Use any method to solve the following resource allocation problem:

$$\begin{aligned} \text{Max } Z &= 20x_1 + 12x_2 + 50x_3 + 25x_4 \dots\dots\dots(\text{profit}) \\ \text{w.r.t.} \\ x_1 + x_2 + x_3 + x_4 &\leq 50 \dots\dots\dots(\text{manpower}) \\ 3x_1 + 2x_2 + x_3 &\leq 100 \dots\dots\dots(\text{material A}) \\ x_2 + 2x_3 &\leq 90, \dots\dots\dots(\text{material B}) \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Source code:

```
from scipy.optimize import linprog

obj = [-20, -12, -40, -25]

In [1]: lhs_ineq = [[1, 1, 1, 1],
                    [3, 2, 1, 0],
                    [0, 1, 2, 3]]

rhs_ineq = [50,
            100,
            90]

In [2]: opt = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
                    method="revised simplex")

opt
```

Output:

```
In [1]: 1 from scipy.optimize import linprog
        2
        3 obj = [-20, -12, -40, -25]
        4
        5 lhs_ineq = [[1, 1, 1, 1],
        6           [3, 2, 1, 0],
        7           [0, 1, 2, 3]]
        8
        9 rhs_ineq = [50,
        10           100,
        11           90]
```

```
In [2]: 1 opt = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq, method="revised simplex")
        2
        3 opt
```

```
Out[2]:      con: array([], dtype=float64)
         fun: -1900.0
         message: 'Optimization terminated successfully.'
         nit: 2
         slack: array([ 0., 40.,  0.])
         status: 0
         success: True
         x: array([ 5.,  0., 45.,  0.])
```

Practical 7

Aim: Use simplex method to solve the following LPP:

$$\text{Max } Z = 200x + 300y$$

w.r.t.

$$2x + 3y \geq 1200,$$

$$x + y \leq 400,$$

$$2x + 1.5y \geq 900,$$

$$x, y \geq 0$$

Source code:

```
from scipy.optimize import linprog

obj = [-200, 300]

lhs_ineq = [[-2, -3],
            [1, 1],
            [-2, -3/2]]
In [1]:

rhs_ineq = [-1200, 400, -900]

bnd=[(0, float("inf")),
      (0, float("inf"))]

opt = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
              bounds = bnd,
              method = "revised simplex")
In [2]:

opt
```

Output:

```
In [1]: 1 from scipy.optimize import linprog
        2
        3 obj = [-200, 300]
        4
        5 lhs_ineq = [[-2, -3],
        6             [1, 1],
        7             [-2, -3/2]]
        8
        9 rhs_ineq = [-1200, 400, -900]
       10
       11 bnd=[(0, float("inf")),
       12        (0, float("inf"))]
```

```
In [2]: 1 opt = linprog(c = obj, A_ub = lhs_ineq, b_ub = rhs_ineq,
        2             bounds = bnd,
        3             method = "revised simplex")
        4
        5 opt
```

```
Out[2]: con: array([], dtype=float64)
        fun: 12000.0
        message: 'The problem appears infeasible, as the phase one auxiliary problem terminated successfully with a residual of
        3.0e+02, greater than the tolerance 1e-12 required for the solution to be considered feasible. Consider increasing the tol
        erance to be greater than 3.0e+02. If this tolerance is unacceptably large, the problem is likely infeasible.'
        nit: 1
        slack: array([ 0.,  0., -300.])
        status: 2
        success: False
        x: array([ 0., 400.])
```


Practical 8

Aim: Use dual simplex method to solve the following LPP:

$$\text{Max } Z = 40x_1 + 50x_2$$

w.r.t.

$$2x_1 + 3x_2 \leq 3,$$

$$8x_1 + 4x_2 \leq 5,$$

$$x_1, x_2 \geq 0$$

Source code:

```
require(lpSolve)
f.obj <- c(40, 50)
f.con <- matrix(c(2, 3,
                  8, 4), nrow = 2, byrow = T)
f.dir <- c("<=", "<=")
f.rhs <- c(3, 5)

lp("max", f.obj, f.con, f.dir, f.rhs)$solution
lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$sens.coef.from
lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$sens.coef.to
lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$duals
lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$duals.to
```

Output:

```
> # Maximize the following:
> # z = 40x1 + 50x2
> # w.r.t
> # 2x1 + 3x2 ≤ 3
> # 8x1 + 4x2 ≤ 5
> # x1, x2 ≥ 0
>
> require(lpSolve)
Loading required package: lpSolve
> f.obj ← c(40, 50)
> f.con ← matrix(c(2, 3,
+                 8, 4), nrow = 2, byrow = T)
> f.dir ← c("≤", "≤")
> f.rhs ← c(3, 5)
>
> lp("max", f.obj, f.con, f.dir, f.rhs)$solution
[1] 0.1875 0.8750
> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$sens.coef.from
[1] 33.33333 20.00000
> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$sens.coef.to
[1] 100 60
```

```
> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$duals
[1] 15.00 1.25 0.00 0.00
> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens = T)$duals.to
[1] 3.75e+00 1.20e+01 1.00e+30 1.00e+30
> |
```

Practical 9

Aim: Solve following transportation problem in which each cell represents unit costs:

	Customers				Supply
	1	2	3	4	
Suppliers	10	2	20	11	15
	12	7	9	20	25
	4	14	16	18	10
Demand	5	15	15	15	

Source code:

```
library(lpSolve)
cost <- matrix(c(10, 2, 20, 11,
                 12, 7, 9, 20,
                 4, 14, 16, 18), nrow = 3, byrow = T)

colnames(cost) <- c("Customer 1", "Customer 2", "Customer 3", "Customer 4")
rownames(cost) <- c("Supplier 1", "Supplier 2", "Supplier 3")

row.signs <- rep("<=", 3)
row.rhs <- c(15, 25, 10)

col.signs <- rep(">=", 4)
col.rhs <- c(5, 15, 15, 15)

total.cost <- lp.transport(cost, "min", row.signs, row.rhs, col.signs,
col.rhs)
total.cost$solution
print(total.cost)
```

Output:

```
> ##SOLVE FOLLOWING TRANSPORTATION PROBLEM IN WHICH CELL ENTRIES REPRESENT UNIT COSTS USING R PROGRAMMING.
>
> #           "Customer 1", "Customer 2", "Customer 3", "Customer 4"  SUPPLY
> #SUPPLIER 1      10         2         20         11         15
> #SUPPLIER 2      12         7         9         20         25
> #SUPPLIER 3       4         14        16         18         10
> #DEMAND         5         15         15         15
>
> library(lpSolve)
> cost ← matrix(c(10, 2, 20, 11,
+                12, 7, 9, 20,
+                4, 14, 16, 18), nrow = 3, byrow = T)
>
> colnames(cost) ← c("Customer 1", "Customer 2", "Customer 3", "Customer 4")
> rownames(cost) ← c("Supplier 1", "Supplier 2", "Supplier 3")
>
> row.signs ← rep("≤", 3)
> row.rhs ← c(15, 25, 10)
>
> col.signs ← rep("≥", 4)
> col.rhs ← c(5, 15, 15, 15)
```

```
> total.cost ← lp.transport(cost, "min", row.signs, row.rhs, col.signs, col.rhs)
> total.cost$solution
      [,1] [,2] [,3] [,4]
[1,]    0    5    0   10
[2,]    0   10   15    0
[3,]    5    0    0    5
> print(total.cost)
Success: the objective function is 435
```

Practical 10

Aim: Solve following assignment problem represented in this matrix:

		Jobs		
		1	2	3
Workers	1	15	10	9
	2	9	15	10
	3	10	12	8

Source Code:

```
library(lpSolve)

cost <- matrix(c(15, 10, 9,
                 9, 15, 10,
                 10, 12, 8), nrow = 3, byrow = T)

cost
answer <- lp.assign(cost)
answer$solution
```

Output:

```
> #SOLVE FOLLOWING ASSIGNMENT PROBLEM REPRESENTED IN FOLLOWING MATRIX USING R PROGRAMMING
> # Assignment Problem
> #      JOB1    JOB2    JOB3
> #W1     15     10     9
> #W2     9      15     10
> #W3     10     12     8
>
> library(lpSolve)
>
> cost <- matrix(c(15, 10, 9,
+                 9, 15, 10,
+                 10, 12, 8), nrow = 3, byrow = T)
>
> cost
      [,1] [,2] [,3]
[1,]   15   10   9
[2,]    9   15  10
[3,]   10   12   8
> answer <- lp.assign(cost)
```

```
> answer ← lp.assign(cost)
> answer$solution
      [,1] [,2] [,3]
[1,]    0    1    0
[2,]    1    0    0
[3,]    0    0    1
> |
```