To solve linear programming using R studio, we need to install lpsolve package Install.packages(“lpsolve”)

**PRACTICAL 1**

**GRAPHICAL METHOD USING R PROGRAMMING**

*# R Program*

*#Find a geometrical interpretation and solution as well for the following LP problem #Max z= 3x1 + 5x2*

*#subject to constraints:*

*#x1+2x2<=2000*

*#x1+x2<=1500*

*#x2<=600*

*#x1,x2>=0*

# Load lpSolve

require(lpSolve)

## Set the coefficients of the decision variables -> C of objective function C <- c(3,5)

# Create constraint martix B

A <- matrix(c(1, 2,

1, 1,

0, 1

), nrow=3, byrow=TRUE)

# Right hand side for the constraints

B <- c(2000,1500,600)

# Direction of the constraints

sconstranints\_direction <- c("<=", "<=", "<=")

# Create empty example plot

plot.new()

plot.window(xlim=c(0,2000), ylim=c(0,2000))

axis(1)

axis(2)

title(main="LPP using Graphical method")

title(xlab="X axis")

title(ylab="Y axis")

box()

# Draw one line

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

# Find the optimal solution

optimum <- lp(direction="max",

objective.in = C,

const.mat = A,

const.dir = constranints\_direction,

const.rhs = B,

all.int = T)

# Print status: 0 = success, 2 = no feasible solution print(optimum$status)

# Display the optimum values for x1,x2

best\_sol <- optimum$solution

names(best\_sol) <- c("x1", "x2")

print(best\_sol)

# Check the value of objective function at optimal point

print(paste("Total cost: ", optimum$objval, sep=""))

OUTPUT:

[Workspace loaded from ~/.RData]

> # Right hand side for the constraints

> B <- c(2000,1500,600)

> # R Program

> # Load lpSolve

> require(lpSolve)

Loading required package: lpSolve

> ## Set the coefficients of the decision variables -> C > C <- c(3,5)

> # Create constraint martix B

> A <- matrix(c(1, 2,

+ 1, 1,

+ 0, 1

+ ), nrow=3, byrow=TRUE)

>

> # Right hand side for the constraints

> B <- c(2000,1500,600)

>

> # Direction of the constraints

> constranints\_direction <- c("<=", "<=", "<=")

>

>

> # Create empty example plot

> #plot(2000, 2000, col = "white", xlab = "", ylab = "") > plot.new()

> plot.window(xlim=c(0,2000), ylim=c(0,2000))

> axis(1)

> axis(2)

> title(main="LPP using Graphical method")

> title(xlab="X axis")

> title(ylab="Y axis")

> box()

> # Draw one line

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

>

>

> # Find the optimal solution

> optimum <- lp(direction="max",

+ objective.in = C,

+ const.mat = A,

+ const.dir = constranints\_direction, + const.rhs = B,

+ all.int = T)

> # Print status: 0 = success, 2 = no feasible solution

> print(optimum$status)

[1] 0

> # Display the optimum values for x1,x2

> best\_sol <- optimum$solution

> names(best\_sol) <- c("x1", "x2")

> print(best\_sol)

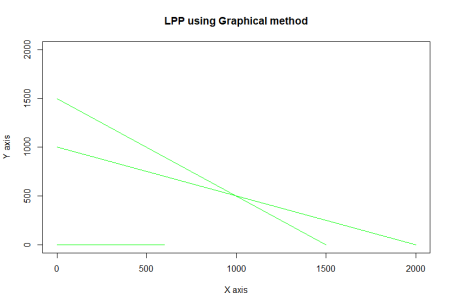
x1 x2

1000 500

>

> # Check the value of objective function at optimal point > print(paste("Total cost: ", optimum$objval, sep=""))

[1] "Total cost: 5500"



**PRACTICAL 2**

**Simplex Method with 2 variables using Python** from scipy.optimize import linprog

#Max z=3x1+2x2

#subject to

#x1 + x2 <=4

#x1 - x2 <=2

#x1,x2>=0

obj = [-3, -2]

lhs\_ineq = [[ 1, 1], # Red constraint left side

... [1, -1]] # Blue constraint left side

rhs\_ineq = [4, # Red constraint right side

... 2] # Blue constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf"))] # Bounds of y

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

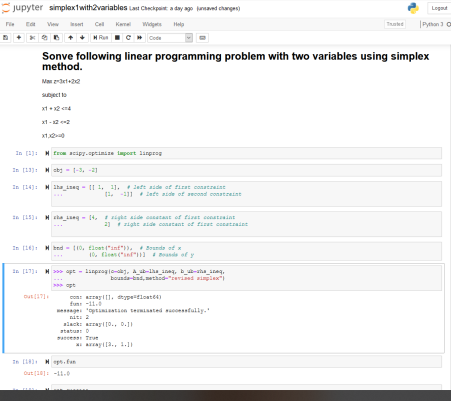
... bounds=bnd,method="revised simplex")

>>> opt

opt.fun

opt.success

opt.x



**PRACTICAL 3**

**Simplex Method with 3 variables using Python** from scipy.optimize import linprog

#Min z= x1-3x2+2x3

#subject to

#3x1-x2+3x3<=7

#-2x1+4x2<=12

#-4x1+3x2+8x3<=10

#x1,x2,x3>=0

obj = [1, -3, 2]

lhs\_ineq = [[ 3, -1, 3], # Red constraint left side

... [-2, 4, 0], # Blue constraint left side

... [ -4, 3, 8]] # Yellow constraint left side

rhs\_ineq = [7, # Red constraint right side

... 12, # Blue constraint right side

... 10] # Yellow constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf")),

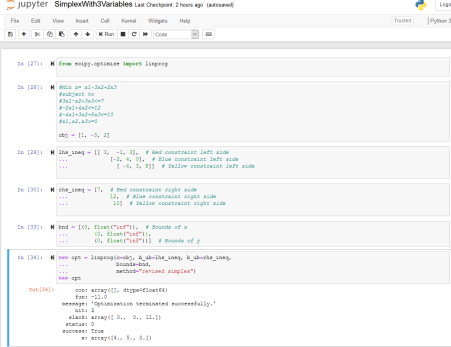
... (0, float("inf"))] # Bounds of y

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... bounds=bnd,

... method="revised simplex")

>>> opt



**PRACTICAL 4**

**Simplex Method with Equality Constraints Using Python**

from scipy.optimize import linprog

#Max z=x+2y

#subject to

#2x+y<=20

#-4x+5y<=10

#-x+2y>=-2

#-x+5y=15

#x,y>=0

obj = [-1, -2]

lhs\_ineq = [[ 2, 1], # Red constraint left side

... [-4, 5], # Blue constraint left side

... [ 1, -2]] # Yellow constraint left side

rhs\_ineq = [20, # Red constraint right side

... 10, # Blue constraint right side

... 2] # Yellow constraint right side

lhs\_eq = [[-1, 5]] # Green constraint left side

rhs\_eq = [15] # Green constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf"))] # Bounds of y

opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... A\_eq=lhs\_eq, b\_eq=rhs\_eq, bounds=bnd,

... method="revised simplex")

Opt

**## method =”revised simplex” solves linear programming problem using two phase simplex method.**

:

con: array([0.])

fun: -16.818181818181817

message: 'Optimization terminated successfully.'

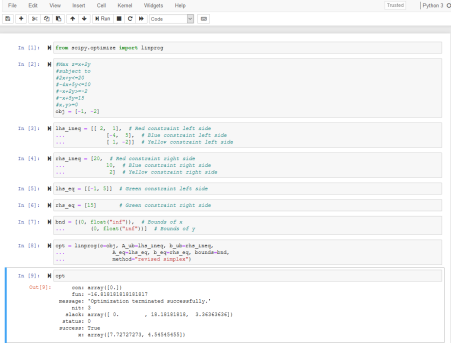
nit: 3

slack: array([ 0. , 18.18181818, 3.36363636])

status: 0

success: True

x: array([7.72727273, 4.54545455])



**PRACTICAL 5**

**BigM Simplex Method using Python**

**Solve Following linear programming problem using Big M Simplex method.** Min z= 4x1 + x2

subjected to:

3x1 + 4x2 >= 20

x1 + 5x2 >= 15

x1, x2 >= 0

from scipy.optimize import linprog

obj = [4, 1]

lhs\_ineq = [[ -3, -4], # left side of first constraint

... [-1, -5]] # right side of first constraint

rhs\_ineq = [-20, # right side of first constraint

... -15] # right side of Second constraint

bnd = [(0, float("inf")), # Bounds of x1

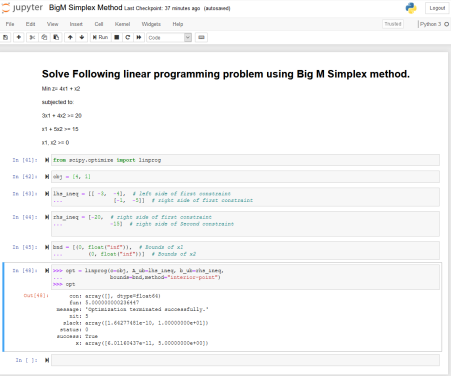
... (0, float("inf"))] # Bounds of x2

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... bounds=bnd,method="interior-point")

>>> opt

**## method =”** interior-point**” solves linear programming problem using default simplex method.**

****

**PRACTICAL 6**

**RESOURCE ALLOCATION PROBLEM BY SIMPLEX METHOD**

**Use SciPy to solve the resource allocation problem stated as follows:** Max z= 20x1 + 12x2 +40x3 + 25x4 .............(profit)

subjected to:

x1 + x2 + x3 + x4 <= 50 -------------(manpower)

3x1 + 2x2 + x3 <= 100 -------------(material A) x2 + 2x3 <= 90 -------------(material B)

x1, x2, x3, x4 >= 0

**from scipy.optimize import linprog**

**obj = [-20, -12, -40, -25] #profit objective function**

**lhs\_ineq = [[1, 1, 1, 1], # Manpower**

**... [3, 2, 1, 0], # Material A**

**... [0, 1, 2, 3]] # Material B**

**rhs\_ineq = [ 50, # Manpower**

**... 100, # Material A**

**... 90] # Material B**

**opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,**

**... method="revised simplex")**

**Opt**

****

**PRACTICAL 7**

**INFEASIBILITY IN SIMPLEX METHOD**

**# Solve following linear programming problem using Simplex method**

**WHILE SOLVING LINEAR PROGRAMMING PROBLEM USING SIMPLEX METHOD, IF ONE OR MORE ARTIFICIAL VARIABLES REMAIN IN THE BASIS AT POSITIVE LEVEL AT THE END OF PHASE 1 COMPUTATION , THE PROBLEM HAS NO FEASIBLE SOLUTION( INFEASIBLE SOLUTION).**

**Example:**

**Max z= 200x - 300y**

**subject to**

**2x+3y>=1200**

**x+y<=400**

**2x+3/2y>=900**

**x,y>=0**

**from scipy.optimize import linprog**

**obj = [-200, 300]**

**lhs\_ineq = [[ -2, -3], # Red constraint left side**

**... [1, 1], # Blue constraint left side**

**... [ -2, -1.5]] # Yellow constraint left side**

**rhs\_ineq = [-1200, # Red constraint right side**

**... 400, # Blue constraint right side**

**... -900] # Yellow constraint right side**

**bnd = [(0, float("inf")), # Bounds of x**

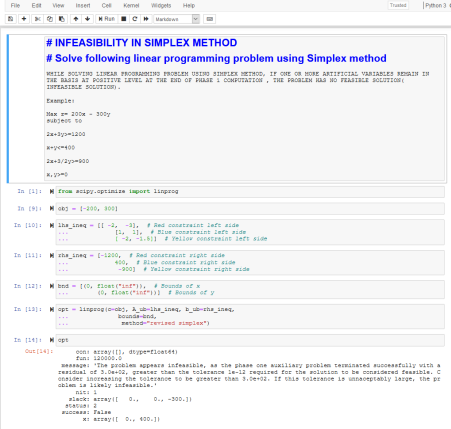
**... (0, float("inf"))] # Bounds of y**

**opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,**

**... bounds=bnd,**

**... method="revised simplex")**

**opt**

****

**PRACTICAL 8**

**DUAL SIMPLEX METHOD**

##SOLVE FOLLOWING LINEAR PROGRAMMING PROBLEM USING DUAL SIMPLEX METHOD USING R PROGRAMMING

# Max z=40x1+50x2

#subject to

#2x1 + 3x2 <= 3

#8x1 + 4x2 <= 5

# x1, x2>=0

# Import lpSolve package

library(lpSolve)

# Set coefficients of the objective function

f.obj <- c(40, 50)

# Set matrix corresponding to coefficients of constraints by rows # Do not consider the non-negative constraint; it is automatically assumed f.con <- matrix(c(2, 3,

8, 4), nrow = 2, byrow = TRUE)

# Set unequality signs

f.dir <- c("<=",

"<=")

# Set right hand side coefficients

f.rhs <- c(3,

5)

# Final value (z)

lp("max", f.obj, f.con, f.dir, f.rhs)

# Variables final values

lp("max", f.obj, f.con, f.dir, f.rhs)$solution

# Sensitivities

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.from lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.to

# Dual Values (first dual of the constraints and then dual of the variables) # Duals of the constraints and variables are mixed

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals

# Duals lower and upper limits

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.from lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.to

**OUTPUT:**

##SOLVE FOLLOWING LINEAR PROGRAMMING PROBLEM USING DUAL SIMPLEX METHOD USING R PROGRAMM> # Max z=40x1+50x2

> #subject to

> #2x1 + 3x2 <= 3

> #8x1 + 4x2 <= 5

> # x1, x2>=0

>

>

> # Import lpSolve package

> library(lpSolve)

>

> # Set coefficients of the objective function

> f.obj <- c(40, 50)

>

> # Set matrix corresponding to coefficients of constraints by rows > # Do not consider the non-negative constraint; it is automatically assumed > f.con <- matrix(c(2, 3,

+ 8, 4), nrow = 2, byrow = TRUE)

>

> # Set unequality signs

> f.dir <- c("<=",

+ "<=")

>

> # Set right hand side coefficients

> f.rhs <- c(3,

+ 5)

>

> # Final value (z)

> lp("max", f.obj, f.con, f.dir, f.rhs)

Success: the objective function is 51.25

>

> # Variables final values

> lp("max", f.obj, f.con, f.dir, f.rhs)$solution

[1] 0.1875 0.8750

>

> # Sensitivities

> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.from [1] 33.33333 20.00000

> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.to [1] 100 60

>

> # Dual Values (first dual of the constraints and then dual of the variables) > # Duals of the constraints and variables are mixed

> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals

[1] 15.00 1.25 0.00 0.00

>

> # Duals lower and upper limits

> lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.from [1] 1.25e+00 4.00e+00 -1.00e+30 -1.00e+30

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

>

**PRACTICAL 9**

**TRANSPORTATION PROBLEM**

##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 1 12 7 9 20 25

#sUPPLIER 1 4 14 16 18 10

#DEMAND 5 15 15 15

# Import lpSolve package

library(lpSolve)

# Set transportation costs matrix

costs <- matrix(c(10, 2, 20, 11,

12, 7, 9, 20,

4, 14 , 16, 18), nrow = 3, byrow = TRUE)

# Set customers and suppliers' names

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

# Set unequality/equality signs for suppliers

row.signs <- rep("<=", 3)

# Set right hand side coefficients for suppliers

row.rhs <- c(15, 25, 10)

# Set unequality/equality signs for customers

col.signs <- rep(">=", 4)

# Set right hand side coefficients for customers

col.rhs <- c(5, 15, 15, 15)

# Final value (z)

TotalCost <- lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)

# Variables final values

lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solution print(TotalCost)

**OUTPUT:**

> ##sOLVE FOLLOWING TRANSPORTATION PROBLEM IN WHICH CELL ENTRIES REPRESENT UNIT COSTS U>

> # "Customer 1", "Customer 2", "Customer 3", "Customer 4" SUPPLY > #sUPPLIER 1 10 2 20 11 15 > #sUPPLIER 1 12 7 9 20 25 > #sUPPLIER 1 4 14 16 18 10 > #DEMAND 5 15 15 15

>

> # Import lpSolve package

> library(lpSolve)

>

> # Set transportation costs matrix

> costs <- matrix(c(10, 2, 20, 11,

+ 12, 7, 9, 20,

+ 4, 14 , 16, 18), nrow = 3, byrow = TRUE)

>

> # Set customers and suppliers' names

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

>

> # Set unequality/equality signs for suppliers

> row.signs <- rep("<=", 3)

>

> # Set right hand side coefficients for suppliers

> row.rhs <- c(15, 25, 10)

>

> # Set unequality/equality signs for customers

> col.signs <- rep(">=", 4)

>

> # Set right hand side coefficients for customers

> col.rhs <- c(5, 15, 15, 15)

>

> # Final value (z)

> TotalCost <- lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs) >

>

> # Variables final values

> lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solution [,1] [,2] [,3] [,4]

[1,] 0 5 0 10

[2,] 0 10 15 0

[3,] 5 0 0 5

>

> print(TotalCost)

Success: the objective function is 435

>

**PRACTICAL 10**

**ASSIGNMENT PROBLEM**

**#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**

**# Import lpSolve package**

**library(lpSolve)**

**# Set assignment costs matrix**

**costs <- matrix(c(15, 10, 9,**

**9, 15, 10,**

**10, 12 ,8), nrow = 3, byrow = TRUE)**

**# Print assignment costs matrix**

**costs**

**# Final value (z)**

**lp.assign(costs)**

**# Variables final values**

**lp.assign(costs)$solution**

**OUTPUT:**

> #SOLVE FOLLOWING ASSIGNMENT PROBLEM REPRESENTED IN FOLLOWING MATRIX USING R PROGRAMMI> # Assignment Problem

> # JOB1 JOB2 JOB3

> #W1 15 10 9

> #W2 9 15 10

> #W3 10 12 8

>

> # Import lpSolve package

> library(lpSolve)

>

> # Set assignment costs matrix

> costs <- matrix(c(15, 10, 9,

+ 9, 15, 10,

+ 10, 12 ,8), nrow = 3, byrow = TRUE) >

> # Print assignment costs matrix

> costs

[,1] [,2] [,3]

[1,] 15 10 9

[2,] 9 15 10

[3,] 10 12 8

>

> # Final value (z)

> lp.assign(costs)

Success: the objective function is 27

>

> # Variables final values

> lp.assign(costs)$solution

[,1] [,2] [,3]

[1,] 0 1 0

[2,] 1 0 0

[3,] 0 0 1